FreeBSD Handbook

Abstract

Welcome to FreeBSD! This handbook covers the installation and day to day use of *FreeBSD 15.0-RELEASE*, *FreeBSD 14.4-RELEASE* and *13.5-RELEASE*. This book is the result of ongoing work by many individuals. Some sections might be outdated. Those interested in helping to update and expand this document should send email to the FreeBSD documentation project mailing list.

The latest version of this book is available from the FreeBSD web site. Previous versions can be obtained from https://docs.FreeBSD.org/doc/. The book can be downloaded in a variety of formats and compression options from the FreeBSD download server or one of the numerous mirror sites. Searches can be performed on the handbook and other documents on the search page.

Table of Contents

Preface 10
- Intended Audience 10
- Fourth Edition 10
- Third Edition 11
- Second Edition (2004) 11
- First Edition (2001) 12
- Organization of This Book 13
- Conventions used in this book 16
- Acknowledgments 17

I: Getting Started 19
- 1. Introduction 20
  - 1.1. Synopsis 20
  - 1.2. Welcome to FreeBSD! 20
  - 1.3. About the FreeBSD Project 22
- 2. Installing FreeBSD 27
  - 2.1. Synopsis 27
  - 2.2. Minimum Hardware Requirements 27
  - 2.3. Pre-Installation Tasks 27
  - 2.4. Starting the Installation 31
  - 2.5. Using bsdinstall 34
  - 2.6. Allocating Disk Space 39
  - 2.7. Fetching Distribution Files 54
  - 2.8. Network Interfaces, Accounts, Time Zone, Services and Hardening 56
  - 2.9. Troubleshooting 79
  - 2.10. Using the Live CD 79
- 3. FreeBSD Basics 81
  - 3.1. Synopsis 81
  - 3.2. Virtual Consoles and Terminals 81
  - 3.3. Users and Basic Account Management 84
  - 3.4. Permissions 93
  - 3.5. Directory Structure 98
  - 3.6. Disk Organization 100
  - 3.7. Mounting and Unmounting File Systems 106
  - 3.8. Processes and Daemons 108
  - 3.9. Shells 111
  - 3.10. Text Editors 114
  - 3.11. Devices and Device Nodes 114
  - 3.12. Manual Pages 114

4. Installing Applications: Packages and Ports 117
  4.1. Synopsis 117
  4.2. Overview of Software Installation 117
  4.3. Finding Software 118
  4.4. Using pkg for Binary Package Management 119
  4.5. Using the Ports Collection 128
  4.6. Building Packages with poudriere 137
  4.7. Post-Installation Considerations 140
  4.8. Dealing with Broken Ports 140
5. The X Window System 142
  5.1. Synopsis 142
  5.2. Graphics Drivers 142
  5.3. X Window System Overview 146
  5.4. Installing The X.org Server 146
  5.5. X.org Configuration 146
  5.6. Using Fonts in the X Window System 152
6. Wayland on FreeBSD 158
  6.1. Synopsis 158
  6.2. Wayland Overview 158
  6.3. The Wayfire Compositor 160
  6.4. The Hikari Compositor 162
  6.5. The Sway Compositor 164
  6.6. Using Xwayland 165
  6.7. Remote Desktop Using VNC 168
  6.8. Wayland Login Manager 168
  6.9. Useful Utilities 169
7. Network 171
  7.1. Synopsis 171
  7.2. Setting up the Network 171
  7.3. Wired Networks 172
  7.4. Wireless Networks 180
  7.5. Hostname 184
  7.6. DNS 185
  7.7. Troubleshooting 186
II: Common Tasks 188
8. Desktop Environments 189
  8.1. Synopsis 189
  8.2. Desktop Environments 189
  8.3. Browsers 196
  8.4. Development tools 199
  8.5. Desktop office productivity 201

8.6. Document Viewers 203
8.7. Finance 204
9. Multimedia 206
9.1. Synopsis 206
9.2. Setting Up the Sound Card 206
9.3. Audio players 210
9.4. Video players 211
9.5. Conferencing and Meetings 213
9.6. Image Scanners 216
10. Configuring the FreeBSD Kernel 220
10.1. Synopsis 220
10.2. Why Build a Custom Kernel? 220
10.3. Finding the System Hardware 221
10.4. The Configuration File 222
10.5. Building and Installing a Custom Kernel 224
10.6. If Something Goes Wrong 225
11. Printing 227
11.1. Quick Start 227
11.2. Printer Connections 228
11.3. Common Page Description Languages 229
11.4. Direct Printing 231
11.5. LPD (Line Printer Daemon) 231
11.6. Other Printing Systems 240
12. Linux Binary Compatibility 241
12.1. Synopsis 241
12.2. Configuring Linux Binary Compatibility 241
12.3. Linux userlands 242
12.4. Advanced Topics 246
13. WINE 250
13.1. Synopsis 250
13.2. WINE Overview & Concepts 251
13.3. Installing WINE on FreeBSD 253
13.4. Running a First WINE Program on FreeBSD 255
13.5. Configuring WINE Installation 256
13.6. WINE Management GUIs 264
13.7. WINE in Multi-User FreeBSD Installations 278
13.8. WINE on FreeBSD FAQ 280
III: System Administration 284
14. Configuration, Services, Logging and Power Management 285
14.1. Synopsis 285
14.2. Configuration Files 285

- 14.3. Managing Services in FreeBSD 290
- 14.4. Cron and Periodic 292
- 14.5. Configuring System Logging 296
- 14.6. Power and Resource Management 305
- 14.7. Adding Swap Space 311

15. The FreeBSD Booting Process 313
- 15.1. Synopsis 313
- 15.2. FreeBSD Boot Process 313
- 15.3. Device Hints 319
- 15.4. Shutdown Sequence 320

16. Security 321
- 16.1. Synopsis 321
- 16.2. Introduction 321
- 16.3. Securing Accounts 322
- 16.4. Intrusion Detection System (IDS) 328
- 16.5. Secure levels 330
- 16.6. File flags 332
- 16.7. OpenSSH 333
- 16.8. OpenSSL 338
- 16.9. Kerberos 342
- 16.10. TCP Wrappers 350
- 16.11. Access Control Lists 351
- 16.12. Capsicum 353
- 16.13. Process Accounting 353
- 16.14. Resource Limits 354
- 16.15. Monitoring Third Party Security Issues 357
- 16.16. FreeBSD Security Advisories 358

17. Jails and Containers 364
- 17.1. Synopsis 364
- 17.2. Jail Types 364
- 17.3. Host Configuration 367
- 17.4. Classic Jail (Thick Jail) 370
- 17.5. Thin Jails 372
- 17.6. Jail Management 381
- 17.7. Jail Upgrading 383
- 17.8. Jail Resource Limits 385
- 17.9. Jail Managers and Containers 386

18. OCI Containers 387
- 18.1. Synopsis 387
- 18.2. Container Image Naming and Tagging 387

19. Mandatory Access Control 391

- 19.1. Synopsis 391
- 19.2. Key Terms 392
- 19.3. Understanding MAC Labels 393
- 19.4. Planning the Security Configuration 397
- 19.5. Available MAC Policies 398
- 19.6. User Lock Down 405
- 19.7. Nagios in a MAC Jail 406
- 19.8. Troubleshooting the MAC Framework 409

20. Security Event Auditing 411
- 20.1. Synopsis 411
- 20.2. Key Terms 411
- 20.3. Audit Configuration 412
- 20.4. Working with Audit Trails 416

21. Storage 420
- 21.1. Synopsis 420
- 21.2. Adding Disks 420
- 21.3. Resizing and Growing Disks 421
- 21.4. USB Storage Devices 424
- 21.5. Creating and Using CD Media 427
- 21.6. Creating and Using DVD Media 433
- 21.7. Creating and Using Floppy Disks 438
- 21.8. Backup Basics 439
- 21.9. Memory Disks 443
- 21.10. File System Snapshots 445
- 21.11. Disk Quotas 447
- 21.12. Encrypting Disk Partitions 450
- 21.13. Encrypting Swap 456
- 21.14. Highly Available Storage (HAST) 458

22. GEOM: Modular Disk Transformation Framework 466
- 22.1. Synopsis 466
- 22.2. RAID0 - Striping 466
- 22.3. RAID1 - Mirroring 468
- 22.4. RAID3 - Byte-level Striping with Dedicated Parity 478
- 22.5. Software RAID Devices 479
- 22.6. GEOM Gate Network 484
- 22.7. Labeling Disk Devices 484
- 22.8. UFS Journaling Through GEOM 487

23. The Z File System (ZFS) 489
- 23.1. What Makes ZFS Different 489
- 23.2. Quick Start Guide 489
- 23.3. `zpool` Administration 496

- 23.4. `zfs` Administration 515
- 23.5. Delegated Administration 536
- 23.6. Advanced Topics 536
- 23.7. Further Resources 539
- 23.8. ZFS Features and Terminology 539

24. Other File Systems 547
- 24.1. Synopsis 547
- 24.2. Linux® File Systems 547
- 24.3. Windows® File Systems 548
- 24.4. MacOS® File Systems 549

25. Virtualization 551
- 25.1. Synopsis 551
- 25.2. FreeBSD as a Guest on Parallels Desktop for macOS® 551
- 25.3. FreeBSD as a Guest on VMware Fusion for macOS® 558
- 25.4. FreeBSD as a Guest on VirtualBox™ 570
- 25.5. FreeBSD as a Host with VirtualBox™ 572
- 25.6. Virtualization with QEMU on FreeBSD 574
- 25.7. FreeBSD as a Host with bhyve 599
- 25.8. FreeBSD as a Xen™-Host 615

26. Localization - i18n/L10n Usage and Setup 622
- 26.1. Synopsis 622
- 26.2. Using Localization 622
- 26.3. Finding i18n Applications 629
- 26.4. Locale Configuration for Specific Languages 629

27. Updating and Upgrading FreeBSD 632
- 27.1. Synopsis 632
- 27.2. FreeBSD Update 632
- 27.3. Updating Bootcode 640
- 27.4. Updating the Documentation Set 640
- 27.5. Tracking a Development Branch 641
- 27.6. Updating FreeBSD from Source 644
- 27.7. Updating FreeBSD with packages 651
- 27.8. Tracking for Multiple Machines 657
- 27.9. Building on non-FreeBSD Hosts 658

28. DTrace 660
- 28.1. Synopsis 660
- 28.2. Implementation Differences 660
- 28.3. Enabling DTrace Support 661
- 28.4. Enabling DTrace in Out-of-Kernel Modules 662
- 28.5. Using DTrace 662

29. USB Device Mode / USB OTG 665

29.1. Synopsis 665
29.2. USB Virtual Serial Ports 665
29.3. USB Device Mode Network Interfaces 667
29.4. USB Virtual Storage Device 668
IV: Network Communication 670
30. Serial Communications 671
30.1. Synopsis 671
30.2. Serial Terminology and Hardware 671
30.3. Terminals 675
30.4. Dial-in Service 679
30.5. Dial-out Service 682
30.6. Setting Up the Serial Console 686
31. PPP 691
31.1. Synopsis 691
31.2. Configuring PPP 691
31.3. Troubleshooting PPP Connections 699
31.4. Using PPP over Ethernet (PPPoE) 702
31.5. Using PPP over ATM (PPPoA) 704
32. Electronic Mail 707
32.1. Synopsis 707
32.2. Mail Components 707
32.3. DragonFly Mail Agent (DMA) 708
32.4. Sendmail 710
32.5. Changing the Mail Transfer Agent 712
32.6. Mail User Agents 715
32.7. Advanced Topics 722
33. Network Servers 728
33.1. Synopsis 728
33.2. The inetd Super-Server 728
33.3. Network File System (NFS) 732
33.4. Network Information System (NIS) 736
33.5. Lightweight Directory Access Protocol (LDAP) 750
33.6. Dynamic Host Configuration Protocol (DHCP) 758
33.7. Domain Name System (DNS) 762
33.8. Zero-configuration Networking (mDNS/DNS-SD) 765
33.9. Apache HTTP Server 765
33.10. File Transfer Protocol (FTP) 772
33.11. File and Print Services for Microsoft® Windows® Clients (Samba) 773
33.12. Clock Synchronization with NTP 776
33.13. iSCSI Initiator and Target Configuration 779
34. Firewalls 785

- 34.1. Synopsis 785
- 34.2. Firewall Concepts 786
- 34.3. PF 787
- 34.4. IPFW 803
- 34.5. IPFILTER (IPF) 819
- 34.6. Blocklistd 831

35. Advanced Networking 836
  - 35.1. Synopsis 836
  - 35.2. Gateways and Routes 836
  - 35.3. Virtual Hosts 842
  - 35.4. Wireless Advanced Authentication 843
  - 35.5. Wireless Ad-hoc Mode 847
  - 35.6. USB Tethering 851
  - 35.7. Bluetooth 852
  - 35.8. Bridging 861
  - 35.9. Link Aggregation and Failover 868
  - 35.10. Diskless Operation with PXE 873
  - 35.11. Common Address Redundancy Protocol (CARP) 878
  - 35.12. VLANs 880

V: Appendices 882

- Appendix A: Obtaining FreeBSD 883
  - A.1. Mirrors 883
  - A.2. Using Git 886
- Appendix B: Bibliography 889
  - B.1. FreeBSD Bibliography 889
  - B.2. Security Reference 889
  - B.3. UNIX® History 889
  - B.4. Periodicals, Journals, and Magazines 890
- Appendix C: Resources on the Internet 891
  - C.1. Websites 891
  - C.2. Mailing Lists 891
  - C.3. Usenet Newsgroups 894
- Appendix D: OpenPGP Keys 895
  - D.1. Officers 895
- FreeBSD Glossary 904
- Colophon 924

Preface

Intended Audience

The FreeBSD newcomer will find that the first section of this book guides the user through the FreeBSD installation process and gently introduces the concepts and conventions that underpin UNIX®. Working through this section requires little more than the desire to explore, and the ability to take on board new concepts as they are introduced.

Once reaching this far, the second, far larger, section of the Handbook is a comprehensive reference to all manner of topics of interest to FreeBSD system administrators. Some of these chapters may recommend to do some prior reading, and this is noted in the synopsis at the beginning of each chapter.

For a list of additional sources of information, please see Bibliography.

Fourth Edition

The current version of the Handbook represents the cumulative effort of a working group that has been reviewing and updating all Handbook content. These are the major updates since the fourth edition of the Handbook.

- The Handbook has been converted from Docbook to Hugo and AsciiDoctor
- The FreeBSD Documentation Portal has been created.
- The Introduction chapter has been updated to improve the FreeBSD history and fix minor typos.
- The Install chapter has been updated with an upgraded synopsis, the latest changes to the installer, refreshed images, added alt text to images, and the removal of references to specific versions.
- The Basics chapter has updated tables, command outputs, and directory structure per man:hier.
- The Ports chapter has been updated and now simplifies package searches, updates software examples (Nginx replaces Apache), improves the pkg(8) bootstrap process, and adds new instructions for configuring and managing packages, including blocking and unblocking.
- The X11 chapter has been updated to reflect the current state of graphics in FreeBSD, remove outdated references to old Intel drivers, configurations, and compiz, and relocate desktop environment instructions (such as KDE Plasma and GNOME) to the Desktop Environments chapter, as these environments now also support Wayland in addition to X11.
- The Wayland chapter has been added with information about installing and configuring Wayland under FreeBSD.
- The Network chapter has been created to cover basic wired and wireless network configuration, including hostname, DNS, and troubleshooting. Sections on wired networks, wireless, and IPv6 have been moved and updated with improved command outputs, use of sysrc, and better AsciiDoc syntax.
- The Desktop chapter has been updated with upgraded installation instructions for KDE Plasma,

GNOME, XFCE, MATE, Cinnamon, and LXQT, expanded browser options, a new development tools section, and updates to office productivity, document viewers, and finance sections.

- The Multimedia chapter has been reworked with updates to the sound section, new tables for sound mixers, audio players, and video players, guidance on automatic headphone switching, a new conferencing and meetings section, and a revised image scanners section.
- The Linuxemu chapter has been improved with updated instructions for setting up a Debian/Ubuntu base system using debootstrap.
- The Config chapter has been renamed for accuracy, with updates to service management, cron and periodic, syslog, power management, and swap sections. A new entry on config files was added, and the outdated tuning section was removed.
- The Security chapter has been updated with enhancements to VPN over IPSec, securing accounts, password hashes, sudo/doas, and OpenSSH/OpenSSL. New sections have been added covering IDS, secure levels, file flags, Capsicum, NFSv4 ACLs, and resource limits.
- The Jails chapter has been updated to include details on jail types (Thick, Thin, VNET, and Linux Jails), host system configuration, networking options, the jail configuration file, setup procedures, upgrade methods, resource limits, and different jail managers and container solutions.
- The Mail chapter has been updated to include information on DMA, upgrades to Sendmail, instructions for changing DMA and Sendmail to use different MTAs, and the removal of Dialup and Fetchmail sections, along with a reorganization of the chapter.
- The Bibliography has been extensively updated.

Third Edition

The current online version of the Handbook represents the cumulative effort of many hundreds of contributors over the past 10 years. The following are some of the significant changes since the two volume third edition was published in 2004:

- WINE has been added with information about how to run Windows® applications on FreeBSD.
- DTrace has been added with information about the powerful DTrace performance analysis tool.
- Other File Systems have been added with information about non-native file systems in FreeBSD, such as ZFS from Sun™.
- Security Event Auditing has been added to cover the new auditing capabilities in FreeBSD and explain its use.
- Virtualization has been added with information about installing FreeBSD on virtualization software.
- Installing FreeBSD has been added to cover installation of FreeBSD using the new installation utility, bsdinstall.

Second Edition (2004)

The third edition was the culmination of over two years of work by the dedicated members of the FreeBSD Documentation Project. The printed edition grew to such a size that it was necessary to publish as two separate volumes. The following are the major changes in this new edition:

- Configuration and Tuning has been expanded with new information about the ACPI power and resource management, the `cron` system utility, and more kernel tuning options.
- Security has been expanded with new information about virtual private networks (VPNs), file system access control lists (ACLs), and security advisories.
- Mandatory Access Control is a new chapter with this edition. It explains what MAC is and how this mechanism can be used to secure a FreeBSD system.
- Storage has been expanded with new information about USB storage devices, file system snapshots, file system quotas, file and network backed filesystems, and encrypted disk partitions.
- A troubleshooting section has been added to PPP.
- Electronic Mail has been expanded with new information about using alternative transport agents, SMTP authentication, UUCP, fetchmail, procmail, and other advanced topics.
- Network Servers is all new with this edition. This chapter includes information about setting up the Apache HTTP Server, ftpd, and setting up a server for Microsoft® Windows® clients with Samba. Some sections from Advanced Networking were moved here to improve the presentation.
- Advanced Networking has been expanded with new information about using Bluetooth® devices with FreeBSD, setting up wireless networks, and Asynchronous Transfer Mode (ATM) networking.
- A glossary has been added to provide a central location for the definitions of technical terms used throughout the book.
- A number of aesthetic improvements have been made to the tables and figures throughout the book.

First Edition (2001)

The second edition was the culmination of over two years of work by the dedicated members of the FreeBSD Documentation Project. The following were the major changes in this edition:

- A complete Index has been added.
- All ASCII figures have been replaced by graphical diagrams.
- A standard synopsis has been added to each chapter to give a quick summary of what information the chapter contains, and what the reader is expected to know.
- The content has been logically reorganized into three parts: "Getting Started", "System Administration", and "Appendices".
- FreeBSD Basics has been expanded to contain additional information about processes, daemons, and signals.
- Installing Applications: Packages and Ports has been expanded to contain additional information about binary package management.
- The X Window System has been completely rewritten with an emphasis on using modern

desktop technologies such as KDE and GNOME on XFree86™ 4.X.

- The FreeBSD Booting Process has been expanded.
- Storage has been written from what used to be two separate chapters on "Disks" and "Backups". We feel that the topics are easier to comprehend when presented as a single chapter. A section on RAID (both hardware and software) has also been added.
- Serial Communications has been completely reorganized and updated for FreeBSD 4.X/5.X.
- PPP has been substantially updated.
- Many new sections have been added to Advanced Networking.
- Electronic Mail has been expanded to include more information about configuring sendmail.
- Linux® Binary Compatibility has been expanded to include information about installing Oracle® and SAP® R/3®.
- The following new topics are covered in this second edition:
    - Configuration and Tuning.
    - Multimedia.

Organization of This Book

This book is split into five logically distinct sections. The first section, *Getting Started*, covers the installation and basic usage of FreeBSD. It is expected that the reader will follow these chapters in sequence, possibly skipping chapters covering familiar topics. The second section, *Common Tasks*, covers some frequently used features of FreeBSD. This section, and all subsequent sections, can be read out of order. Each chapter begins with a succinct synopsis that describes what the chapter covers and what the reader is expected to already know. This is meant to allow the casual reader to skip around to find chapters of interest. The third section, *System Administration*, covers administration topics. The fourth section, *Network Communication*, covers networking and server topics. The fifth section contains appendices of reference information.

Introduction

Introduces FreeBSD to a new user. It describes the history of the FreeBSD Project, its goals and development model.

Installing FreeBSD

Walks a user through the entire installation process of FreeBSD 9.*x* and later using bsdinstall.

FreeBSD Basics

Covers the basic commands and functionality of the FreeBSD operating system. When familiar with Linux® or another flavor of UNIX® then this chapter can probably be skipped.

Installing Applications: Packages and Ports

Covers the installation of third-party software with both FreeBSD's innovative "Ports Collection" and standard binary packages.

The X Window System

Describes the X Window System in general and using X11 on FreeBSD in particular. Also describes common desktop environments such as KDE and GNOME.

Wayland

Describes the Wayland display server in general and using Wayland on FreeBSD in particular. Also describes common compositors such as Wayfire, Hikari and Sway.

Desktop Applications

Lists some common desktop applications, such as web browsers and productivity suites, and describes how to install them on FreeBSD.

Multimedia

Shows how to set up sound and video playback support for the system. Also describes some sample audio and video applications.

Configuring the FreeBSD Kernel

Explains the need to configure a new kernel and provides detailed instructions for configuring, building, and installing a custom kernel.

Printing

Describes managing printers on FreeBSD, including information about banner pages, printer accounting, and initial setup.

Linux® Binary Compatibility

Describes the Linux® compatibility features of FreeBSD. Also provides detailed installation instructions for many popular Linux® applications such as Oracle® and Mathematica®.

WINE

Describes WINE and provides detailed installation instructions. Also describes how WINE operates, how to install a GUI helper, how to run Windows® applications on FreeBSD, and offers other tips and solutions.

Configuration and Tuning

Describes the parameters available for system administrators to tune a FreeBSD system for optimum performance. Also describes the various configuration files used in FreeBSD and where to find them.

The FreeBSD Booting Process

Describes the FreeBSD boot process and explains how to control this process with configuration options.

Security

Describes many different tools available to help keep the FreeBSD system secure, including Kerberos, IPsec and OpenSSH.

Jails

Describes the jails framework, and the improvements of jails over the traditional chroot support of FreeBSD.

Mandatory Access Control

Explains what Mandatory Access Control (MAC) is and how this mechanism can be used to secure a FreeBSD system.

Security Event Auditing

Describes what FreeBSD Event Auditing is, how it can be installed, configured, and how audit trails can be inspected or monitored.

Storage

Describes how to manage storage media and filesystems with FreeBSD. This includes physical disks, RAID arrays, optical and tape media, memory-backed disks, and network filesystems.

GEOM: Modular Disk Transformation Framework

Describes what the GEOM framework in FreeBSD is and how to configure various supported RAID levels.

The OpenZFS storage platform

Describes the OpenZFS storage platform and provides a quick-start guide and information about advanced topics running OpenZFS under FreeBSD.

Other File Systems

Examines support for non-native file systems under FreeBSD like ext2, ext3 and ext4.

Virtualization

Describes what virtualization systems offer, and how they can be used with FreeBSD.

Localization - i18n/L10n Usage and Setup

Describes how to use FreeBSD in languages other than English. Covers both system and application level localization.

Updating and Upgrading FreeBSD

Explains the differences between FreeBSD-STABLE, FreeBSD-CURRENT, and FreeBSD releases. Describes which users would benefit from tracking a development system and outlines that process. Covers the methods users may take to update their system to the latest security release.

DTrace

Describes how to configure and use the DTrace tool from Sun™ on FreeBSD. Dynamic tracing can help locate performance issues, by performing real time system analysis.

USB Device Mode / USB OTG

Explains the use of USB Device Mode and USB On The Go (USB OTG) on FreeBSD.

PPP

Describes how to use PPP to connect to remote systems in FreeBSD.

Electronic Mail

Explains the different components of an email server and dives into simple configuration topics for the most popular mail server software: sendmail.

Network Servers

Provides detailed instructions and example configuration files to set up the FreeBSD machine as a network filesystem server, domain name server, network information system server, or time synchronization server.

Firewalls

Explains the philosophy behind software-based firewalls and provides detailed information about the configuration of the different firewalls available for FreeBSD.

Advanced Networking

Describes many networking topics, including sharing an Internet connection with other computers on the LAN, advanced routing topics, wireless networking, Bluetooth®, ATM, IPv6, and much more.

Obtaining FreeBSD

Lists different sources for obtaining FreeBSD media on CDROM or DVD as well as different sites on the Internet that allows to download and install FreeBSD.

Bibliography

This book touches on many different subjects that may leave readers hungry for a more detailed explanation. The bibliography lists many excellent books that are referenced in the text.

Resources on the Internet

Describes the many forums available for FreeBSD users to post questions and engage in technical conversations about FreeBSD.

OpenPGP Keys

Lists the PGP fingerprints of several FreeBSD Developers.

Conventions used in this book

To provide a consistent and easy to read text, several conventions are followed throughout the book.

Typographic Conventions

Italic

An *italic* font is used for filenames, URLs, emphasized text, and the first usage of technical terms.

`Monospace`

A `monospaced` font is used for error messages, commands, environment variables, names of ports, hostnames, user names, group names, device names, variables, and code fragments.

Bold

A **bold** font is used for applications, commands, and keys.

User Input

Keys are shown in **bold** to stand out from other text. Key combinations that are meant to be typed simultaneously are shown with `+` between the keys, such as:

`Ctrl`+`Alt`+`Del`

Meaning the user should type the `Ctrl`, `Alt`, and `Del` keys at the same time.

Keys that are meant to be typed in sequence will be separated with commas, for example:

`Ctrl`+`X`, `Ctrl`+`S`

Would mean that the user is expected to type the `Ctrl` and `X` keys simultaneously and then to type the `Ctrl` and `S` keys simultaneously.

Examples

Examples starting with C:\> indicate a MS-DOS® command. Unless otherwise noted, these commands may be executed from a "Command Prompt" window in a modern Microsoft® Windows® environment.

```
C:\> tools\fdimage floppies\kern.flp A:
```

Examples starting with # indicate a command that must be invoked as the superuser in FreeBSD. Login as `root` to type the command, or login as the normal account and use su(1) to gain superuser privileges.

```
# dd if=kern.flp of=/dev/fd0
```

Examples starting with % indicate a command that should be invoked from a normal user account. Unless otherwise noted, C-shell syntax is used for setting environment variables and other shell commands.

```
% top
```

Acknowledgments

This book represents the efforts of many hundreds of people around the world. Whether they sent in fixes for typos, or submitted complete chapters, all the contributions have been useful.

Several companies have supported the development of this document by paying authors to work on it full-time, paying for publication, etc. In particular, BSDi (subsequently acquired by Wind River Systems) paid members of the FreeBSD Documentation Project to work on improving this book full time leading up to the publication of the first printed edition in March 2000 (ISBN 1-57176-241-8). Wind River Systems then paid several additional authors to make a number of improvements to the

print-output infrastructure and to add additional chapters to the text. This work culminated in the publication of the second printed edition in November 2001 (ISBN 1-57176-303-1). In 2003-2004, FreeBSD Mall Inc. paid several contributors to improve the Handbook in preparation for the third printed edition. The third printed edition has been split into two volumes. Both volumes have been published as The FreeBSD Handbook 3rd Edition Volume 1: User Guide (ISBN 1-57176-327-9) and The FreeBSD Handbook 3rd Edition Volume 2: Administrators Guide (ISBN 1-57176-328-7).

Part I: Getting Started

This part of the handbook is for users and administrators who are new to FreeBSD. These chapters:

- Introduce FreeBSD.
- Guide readers through the installation process.
- Teach UNIX® basics and fundamentals.
- Show how to install the wealth of third party applications available for FreeBSD.
- Introduce X, the UNIX® windowing system, and detail how to configure a desktop environment that makes users more productive.
- Introduce Wayland, a new display server for UNIX®.

The number of forward references in the text have been kept to a minimum so that this section can be read from front to back with minimal page flipping.

Chapter 1. Introduction

1.1. Synopsis

Thank you for your interest in FreeBSD! The following chapter covers various aspects of the FreeBSD Project, such as its history, goals, development model, and so on.

Read this chapter to learn:

- How FreeBSD relates to other computer operating systems.
- The history of the FreeBSD Project.
- The goals of the FreeBSD Project.
- The basics of the FreeBSD open-source development model.
- Where the name "FreeBSD" comes from.

1.2. Welcome to FreeBSD!

FreeBSD is an Open Source, standards-compliant Unix-like operating system for x86 (both 32 and 64 bit), ARM, AArch64, RISC-V, POWER, and PowerPC computers. It provides all the features that are nowadays taken for granted, such as preemptive multitasking, memory protection, virtual memory, multi-user facilities, SMP support, all the Open Source development tools for different languages and frameworks, and desktop features centered around X Window System, KDE, or GNOME. Its particular strengths are:

- *Liberal Open Source license*, which grants the rights to freely modify and extend its source code and incorporate it in both Open Source projects and closed products without imposing restrictions typical to copyleft licenses, as well as avoiding potential license incompatibility problems.
- *Strong TCP/IP networking* - FreeBSD implements industry standard protocols with ever increasing performance and scalability. This makes it a good match in both server, and routing/firewalling roles - and indeed many companies and vendors use it precisely for that purpose.
- *Fully integrated OpenZFS support*, including root-on-ZFS, ZFS Boot Environments, fault management, administrative delegation, support for jails, FreeBSD specific documentation, and system installer support.
- *Extensive security features*, from the Mandatory Access Control framework to Capsicum capability and sandbox mechanisms.
- *Over 30 thousand prebuilt packages* for all supported architectures, and the Ports Collection which makes it easy to build customized ones.
- *Documentation* - in addition to the Handbook and books from different authors that cover topics ranging from system administration to kernel internals, there are also the man(1) pages, not only for userspace daemons, utilities, and configuration files, but also for kernel driver APIs (section 9) and individual drivers (section 4).

- *Simple and consistent repository structure and build system* - FreeBSD uses a single repository for all of its components, both kernel and userspace. This, along with a unified and easy to customize build system and a well thought-out development process makes it easy to integrate FreeBSD with build infrastructure for third-party or derived products.
- *Staying true to Unix philosophy*, preferring composability instead of monolithic "all in one" daemons with hardcoded behavior.
- *Binary compatibility* with Linux, which makes it possible to run many Linux binaries without the need for virtualisation.

FreeBSD is based on the 4.4BSD-Lite release from Computer Systems Research Group (CSRG) at the University of California at Berkeley, and carries on the distinguished tradition of BSD systems development. In addition to the fine work provided by CSRG, the FreeBSD Project has put in many thousands of man-hours into extending the functionality and fine-tuning the system for maximum performance and reliability in real-life load situations. FreeBSD offers performance and reliability on par with other Open Source and commercial offerings, combined with cutting-edge features not available anywhere else.

1.2.1. What Can FreeBSD Do?

The applications to which FreeBSD can be put are truly limited only by ones own imagination. From software development to factory automation, inventory control to azimuth correction of remote satellite antenna; if it can be done with a commercial UNIX® product then it is more than likely that FreeBSD can do it, too! FreeBSD also benefits significantly from literally thousands of high quality applications developed by research centers and universities around the world, often available at little to no cost.

Because the source code for FreeBSD itself is freely available, the system can also be customized to an almost unheard-of degree for special applications or projects, and in ways not generally possible with operating systems from most major commercial vendors. Here is just a sampling of some of the applications in which people are currently using FreeBSD:

- *Internet Services:* The robust TCP/IP networking built into FreeBSD makes it an ideal platform for a variety of Internet services such as:
    - Web servers
    - IPv4 and IPv6 routing
    - Firewalls and NAT ("IP masquerading") gateways
    - FTP servers
    - Email servers
    - Storage servers
    - Virtualization servers
    - And more…
- *Education:* Computer science students, as well as those in related engineering fields, will benefit from FreeBSD. There is no better way of learning about operating systems, computer architecture and networking than the hands-on, under-the-hood experience that FreeBSD can

provide. A number of freely available CAD, mathematical and graphic design packages also make it highly useful to those whose primary interest in a computer is to get *other* work done!

- *Research:* With source code for the entire system available, FreeBSD is an excellent platform for research in operating systems as well as other branches of computer science. FreeBSD's freely available nature also makes it possible for remote groups to collaborate on ideas or shared development without having to worry about special licensing agreements or limitations on what may be discussed in open forums.
- *Networking:* Need a new router? A name server (DNS)? A firewall to keep people out of the internal network? FreeBSD can easily turn that unused PC sitting in the corner into an advanced router with sophisticated packet-filtering capabilities.
- *Embedded:* FreeBSD makes an excellent platform to build embedded systems upon. With support for the ARM, AArch64 and PowerPC platforms, coupled with a robust network stack, cutting edge features, and the permissive BSD license, FreeBSD makes an excellent foundation for building embedded routers, firewalls, and other devices.
- *Desktop:* FreeBSD makes a fine choice for an inexpensive desktop solution using the freely available X11 server and Wayland display server. FreeBSD offers a choice from many open-source desktop environments, including the standard GNOME and KDE graphical user interfaces. FreeBSD can even boot "diskless" from a central server, making individual workstations even cheaper and easier to administer.
- *Software Development:* The basic FreeBSD system comes with a full suite of development tools including a full C/C++ compiler and debugger suite. Support for many other languages are also available through the ports and packages collection.

FreeBSD is available to download and in most clouds, free of charge. Please see Obtaining FreeBSD for more information about obtaining FreeBSD.

1.2.2. Who Uses FreeBSD?

FreeBSD has been known for its web serving capabilities. A list of testimonials from companies basing their products and services on FreeBSD can be found at the FreeBSD Foundation website. Wikipedia also maintains a list of products based on FreeBSD.

1.3. About the FreeBSD Project

The following section provides some background information on the project, including a brief history, project goals, and the development model of the project.

1.3.1. A Brief History of FreeBSD

The FreeBSD Project had its genesis in the early part of 1993, partially as the brainchild of the Unofficial 386BSDPatchkit's last 3 coordinators: Nate Williams, Rod Grimes and Jordan Hubbard.

The original goal was to produce an intermediate snapshot of 386BSD in order to fix a number of problems that the patchkit mechanism was just not capable of solving. The early working title for the project was 386BSD 0.5 or 386BSD Interim in reference to that fact.

386BSD was Bill Jolitz's operating system, which had been up to that point suffering rather severely from almost a year's worth of neglect. As the patchkit swelled ever more uncomfortably with each passing day, they decided to assist Bill by providing this interim "cleanup" snapshot. Those plans came to a rude halt when Bill Jolitz suddenly decided to withdraw his sanction from the project without any clear indication of what would be done instead.

The trio thought that the goal remained worthwhile, even without Bill's support, and so they adopted the name "FreeBSD" coined by David Greenman. The initial objectives were set after consulting with the system's current users and, once it became clear that the project was on the road to perhaps even becoming a reality, Jordan contacted Walnut Creek CDROM with an eye toward improving FreeBSD's distribution channels for those many unfortunates without easy access to the Internet. Walnut Creek CDROM not only supported the idea of distributing FreeBSD on CD but also went so far as to provide the project with a machine to work on and a fast Internet connection. Without Walnut Creek CDROM's almost unprecedented degree of faith in what was, at the time, a completely unknown project, it is quite unlikely that FreeBSD would have gotten as far, as fast, as it has today.

The first CD-ROM (and general net-wide) distribution was FreeBSD 1.0, released in December of 1993. This was based on the 4.3BSD-Lite ("Net/2") tape from U.C. Berkeley, with many components also provided by 386BSD and the Free Software Foundation. It was a fairly reasonable success for a first offering, and they followed it with the highly successful FreeBSD 1.1 release in May of 1994.

Around this time, some rather unexpected storm clouds formed on the horizon as Novell and U.C. Berkeley settled their long-running lawsuit over the legal status of the Berkeley Net/2 tape. A condition of that settlement was U.C. Berkeley's concession that three files of Net/2 were "encumbered" code and had to be removed as they were the property of Novell, who had in turn acquired it from AT&T some time previously. What Berkeley got in return was Novell's "blessing" that the 4.4BSD-Lite release, when it was finally released, would be declared unencumbered and all existing Net/2 users would be strongly encouraged to switch. This included FreeBSD, and the project was given until the end of July 1994 to stop shipping its own Net/2 based product. Under the terms of that agreement, the project was allowed one last release before the deadline, that release being FreeBSD 1.1.5.1.

FreeBSD then set about the arduous task of literally re-inventing itself from a completely new and rather incomplete set of 4.4BSD-Lite bits. Although only three files having to do with System V shared memory and semaphores were removed, many other changes and bug fixes had been made to the BSD distribution, so it was a huge task to merge all the FreeBSD developments into 4.4BSD-Lite. It took the project until November of 1994 to make this transition, and in December it released FreeBSD 2.0 to the world. Despite being still more than a little rough around the edges, the release was a significant success and was followed by the more robust and easier to install FreeBSD 2.0.5 release in June of 1995.

Since that time, FreeBSD has made a series of releases each time improving the stability, speed, and feature set of the previous version.

For now, long-term development projects continue to take place in the 16.0-CURRENT (main) branch, and snapshot releases of 16.0 are continually made available from the snapshot server as work progresses.

1.3.2. FreeBSD Project Goals

The goals of the FreeBSD Project are to provide software that may be used for any purpose and without strings attached. Many of us have a significant investment in the code (and project) and would certainly not mind a little financial compensation now and then, but we are definitely not prepared to insist on it. We believe that our first and foremost "mission" is to provide code to any and all comers, and for whatever purpose, so that the code gets the widest possible use and provides the widest possible benefit. This is, we believe, one of the most fundamental goals of Free Software and one that we enthusiastically support.

That code in our source tree which falls under the GNU General Public License (GPL) or Library General Public License (LGPL) comes with slightly more strings attached, though at least on the side of enforced access rather than the usual opposite. Due to the additional complexities that can evolve in the commercial use of GPL software we do, however, prefer software submitted under the more relaxed BSD license when it is a reasonable option to do so.

1.3.3. The FreeBSD Development Model

The development of FreeBSD is a very open and flexible process, being literally built from the contributions of thousands of people around the world, as can be seen from our list of contributors. FreeBSD's development infrastructure allows these thousands of contributors to collaborate over the Internet. We are constantly on the lookout for new volunteers, and those interested in becoming more closely involved should consult the article on Contributing to FreeBSD.

Useful things to know about the FreeBSD Project and its development process, whether working independently or in close cooperation:

The Git repositories

For several years, the central source tree for FreeBSD was maintained by CVS (Concurrent Versions System), a freely available source code control tool. In June 2008, the Project switched to using SVN (Subversion). The switch was deemed necessary, as the technical limitations imposed by CVS were becoming obvious due to the rapid expansion of the source tree and the amount of history already stored. The Documentation Project and Ports Collection repositories also moved from CVS to SVN in May 2012 and July 2012, respectively. In December 2020, the Project migrated Source and Documentation repositories to Git, with Ports following suit in April 2021. Please refer to the Obtaining the Source section for more information on obtaining the FreeBSD `src/` repository and Using the Ports Collection for details on obtaining the FreeBSD Ports Collection.

The committers list

The *committers* are the people who have *push* access to the Git repository, and are authorized to make modifications to the FreeBSD source (the term "committer" comes from `commit`, the source control command which is used to bring new changes into the repository). Anyone can submit a bug to the Bug Database. Before submitting a bug report, the FreeBSD mailing lists, IRC channels, or forums can be used to help verify that an issue is actually a bug.

The FreeBSD core team

The *FreeBSD core team* would be equivalent to the board of directors if the FreeBSD Project were a company. The primary task of the core team is to make sure the project, as a whole, is in good shape and is heading in the right directions. Inviting dedicated and responsible developers to join our group of committers is one of the functions of the core team, as is the recruitment of new core team members as others move on. The current core team was elected from a pool of committer candidates in May and June 2024. Elections are held every 2 years.

> Like most developers, most members of the core team are also volunteers when it comes to FreeBSD development and do not benefit from the project financially, so "commitment" should also not be misconstrued as meaning "guaranteed support." The "board of directors" analogy above is not very accurate, and it may be more suitable to say that these are the people who gave up their lives in favor of FreeBSD against their better judgement!

The FreeBSD Foundation

The FreeBSD Foundation is a 501(c)(3), US-based, non-profit organization dedicated to supporting and promoting the FreeBSD Project and community worldwide. The Foundation funds software development via project grants and provides staff to immediately respond to urgent problems and implement new features and functionality. The Foundation purchases hardware to improve and maintain FreeBSD infrastructure, and funds staffing to improve test coverage, continuous integration and automation. The Foundation advocates for FreeBSD by promoting FreeBSD at technical conferences and events around the world. The Foundation also provides workshops, educational material, and presentations to recruit more users and contributors to FreeBSD. The Foundation also represents the FreeBSD Project in executing contracts, license agreements, and other legal arrangements that require a recognized legal entity.

Outside contributors

Last, but definitely not least, the largest group of developers are the users themselves who provide feedback and bug fixes to us on an almost constant basis. The primary way of keeping in touch with the development of the FreeBSD base system is to subscribe to the FreeBSD technical discussions mailing list where such things are discussed. For porting third party applications, it would be the FreeBSD ports mailing list. For documentation - FreeBSD documentation project mailing list. See Resources on the Internet for more information about the various FreeBSD mailing lists.

The FreeBSD Contributors List is a long and growing one, so why not join it by contributing something back to FreeBSD today? Providing code is not the only way!

In summary, our development model is organized as a loose set of concentric circles. The centralized model is designed for the convenience of the *users* of FreeBSD, who are provided with an easy way of tracking one central code base, not to keep potential contributors out! Our desire is to present a stable operating system with a large set of coherent application programs that the users can easily install and use - this model works very well in accomplishing that.

All we ask of those who would join us as FreeBSD developers is some of the same dedication its current people have to its continued success!

1.3.4. Third Party Programs

In addition to the base distributions, FreeBSD offers a ported software collection with thousands of commonly sought-after programs. The list of ports ranges from HTTP servers to games, languages, editors, and almost everything in between. There are about 36000 ports; the entire Ports Collection requires approximately 3 GB. To compile a port, simply change to the directory of the program needed to install, type `make install`, and let the system do the rest. The full original distribution for each port that is built is retrieved dynamically so only enough disk space to build the port is needed.

Almost every port is also provided as a pre-compiled "package", which can be installed with a simple command (`pkg install`) by those who do not wish to compile their own ports from source. More information on packages and ports can be found in Installing Applications: Packages and Ports.

1.3.5. Additional Documentation

All supported FreeBSD versions provide an option in the installer to install additional documentation under /usr/local/share/doc/freebsd during the initial system setup. Documentation may also be installed later using packages:

```
# pkg install en-freebsd-doc
```

For localized versions replace the "en" with the language prefix of choice. Be aware that some of the localised versions might be out of date and might contain information that is no longer correct or relevant. Read the locally installed manuals with a web browser using the following URLs:

The FreeBSD Handbook

`/usr/local/share/doc/freebsd/en/books/handbook/handbook_en.pdf`

The FreeBSD FAQ

`/usr/local/share/doc/freebsd/en/books/faq/faq_en.pdf`

Find up to date documentation at The Documentation Portal.

All trademarks are the property of their respective owners.

Chapter 2. Installing FreeBSD

2.1. Synopsis

FreeBSD supports different architectures including amd64, ARM®, RISC-V®, and PowerPC®. Depending on the architecture and platform, different images can be downloaded to install or directly run FreeBSD.

The image types are:

- Virtual Machine disk images, such as `qcow2`, `vmdk`, `vhd`, and raw device images. These are not installation images, but images that have FreeBSD preinstalled and ready for post-installation tasks. Virtual machine images are also commonly used in cloud environments.
- SD card images, for embedded systems such as Raspberry Pi. These files must be uncompressed and written as a raw image to an SD card, from which the board will boot.
- Installation images to boot from an ISO or USB device to install FreeBSD on a drive for the usual desktop, laptop, or server system.

The rest of this chapter describes the third case, explaining how to install FreeBSD using the text-based installation program named bsdinstall. There may be minor differences between the installer and what is shown here, so use this chapter as a general guide rather than as a set of literal instructions.

Read this chapter to learn:

- How to obtain FreeBSD images and create FreeBSD installation media.
- How to start bsdinstall.
- The questions bsdinstall will ask, what they mean, and how to answer them.
- How to troubleshoot a failed installation.
- How to access a live version of FreeBSD before committing to an installation.

2.2. Minimum Hardware Requirements

The hardware requirements to install FreeBSD vary by architecture and version. Hardware architectures and devices supported by a FreeBSD release are listed on the FreeBSD Release Information page. The FreeBSD download page also has recommendations for choosing the correct image for different architectures.

2.3. Pre-Installation Tasks

Once it has been determined that the system meets the minimum hardware requirements for installing FreeBSD, the installation file should be downloaded and the installation media prepared.

Consider using virtualization when wanting to use FreeBSD on a system that

> already has another operating system installed.

Before moving on to the installation, check that the system is ready by verifying the items in this checklist:

1. **Back Up Important Data**

   Before installing any operating system, **always** backup all important data first. Do not store the backup on the system being installed. Instead, save the data to a removable disk such as a USB drive, another system on the network, or an online backup service. Test the backup before starting the installation to make sure it contains all of the needed files. Once the installer formats the system's disk, all data stored on that disk will be lost.

2. **Decide Where to Install FreeBSD**

   If FreeBSD will be the only operating system installed, this step can be skipped. But if FreeBSD will share the disk with another operating system, decide which disk or partition will be used for FreeBSD.

   In the i386 and amd64 architectures, disks can be divided into multiple partitions using one of two partitioning schemes. A traditional *Master Boot Record* (MBR) holds a partition table defining up to four *primary partitions*. For historical reasons, FreeBSD calls these primary partition *slices*. One of these primary partitions can be made into an *extended partition* containing multiple *logical partitions*. The *GUID Partition Table* (GPT) is a newer and simpler method of partitioning a disk. Common GPT implementations allow up to 128 partitions per disk, eliminating the need for logical partitions.

   The FreeBSD boot loader requires either a primary or GPT partition. If all of the primary or GPT partitions are already in use, one must be freed for FreeBSD. To create a partition without deleting existing data, use a partition resizing tool to shrink an existing partition and create a new partition using the freed space.

   An alternative to modifying the system's existing disk partitions is to use virtualization, which allows multiple operating systems to run at the same time without having to alter partitions.

   A variety of free and commercial partition resizing tools are listed at List of disk partitioning software wikipedia entry. GParted Live is a free live CD which includes the GParted partition editor.

   > ⚠ When used properly, disk shrinking utilities can safely create space for creating a new partition. Since the possibility of selecting the wrong partition exists, always backup any important data and verify the integrity of the backup before modifying disk partitions.

   Disk partitions containing different operating systems make it possible to install multiple operating systems on one computer.

3. **Collect Network Information**

Some FreeBSD installation methods require a network connection in order to download the installation files. After any installation, the installer will offer to setup the system's network interfaces.

If the network has a DHCP server, it can be used to provide automatic network configuration. If DHCP is not available, the following network information for the system must be obtained from the local network administrator or Internet service provider:

Required Network Information

a. IP address
b. Subnet mask
c. IP address of default gateway
d. Domain name of the network
e. IP addresses of the network's DNS servers

4. **Check for FreeBSD Errata**

   Although the FreeBSD Project strives to ensure that each release of FreeBSD is as stable as possible, bugs occasionally creep into the process. On very rare occasions those bugs affect the installation process. As these problems are discovered and fixed, they are noted in the FreeBSD Errata page of each version. Check the errata before installing to make sure that there are no problems that might affect the installation.

   Information and errata for all the releases can be found on the FreeBSD Release Information page.

2.3.1. Prepare the Installation Media

The FreeBSD installer is not an application that can be run from within another operating system. Instead, download a FreeBSD installation file, burn it to the media associated with its file type and size (CD, DVD, or USB), and boot the system to install from the inserted media.

FreeBSD installation files are available at the FreeBSD download page. Each installation file's name includes the release version of FreeBSD, the architecture, and the type of file.

Installation files are available in several formats, compressed with xz(1) or uncompressed. The formats vary depending on computer architecture and media type.

Installation file types:

- `-bootonly.iso`: This is the smallest installation file as it only contains the installer. A working Internet connection is required during installation as the installer will download the files it needs to complete the FreeBSD installation. This file should be burned to optical media.
- `-disc1.iso`: This file contains all of the files needed to install FreeBSD, its source, and the Ports Collection. This file should be burned to optical media.
- `-dvd1.iso`: This file contains all of the files needed to install FreeBSD, its source, and the Ports

Collection. It also contains a set of popular binary packages for installing a window manager and some applications so that a complete system can be installed from media without requiring a connection to the Internet. This file should be burned to optical media.

- `-memstick.img`: This file contains all of the files needed to install FreeBSD, its source, and the Ports Collection. Write this file to a USB stick as shown in Writing an Image File to USB.
- `-mini-memstick.img`: Like `-bootonly.iso`, does not include installation files, but downloads them as needed. A working internet connection is required during installation. It should be written to a USB stick as shown in Writing an Image File to USB.

After downloading the image file, download at least one *checksum* file from the same directory. There are two *checksum* files available, named after the release number and the architecture name. For example: `CHECKSUM.SHA256-FreeBSD-15.1-RELEASE-amd64` and `CHECKSUM.SHA512-FreeBSD-15.1-RELEASE-amd64`.

After downloading one of the files (or both), calculate the *checksum* for the image file and compare it with the one shown in the *checksum* file. Note that comparing the calculated *checksum* against the correct file is needed, as they correspond to two different algorithms: SHA256 and SHA512. FreeBSD provides sha256(1) and sha512(1) that can be used for calculating the *checksum*. Other operating systems have similar programs.

Verifying the *checksum* in FreeBSD can be done automatically using sha256sum(1) (and sha512sum(1)) by executing:

```
% sha256sum -c CHECKSUM.SHA256-FreeBSD-15.1-RELEASE-amd64 --ignore-missing
FreeBSD-15.1-RELEASE-amd64-dvd1.iso: OK
```

The checksums must match exactly. If the checksums do not match, the image file is corrupt and must be downloaded again.

Writing an Image File to USB

The `*memstick.img` file is an *image* of the complete contents of a memory stick. It *cannot* be copied to the target device as a file. Several applications are available for writing the `*.img` to a USB stick. This section describes two of these utilities.

Before proceeding, back up any important data on the USB stick. This procedure will erase the existing data on the stick.

Procedure. Using `dd` to write the image

This example uses `/dev/da0` as the target device where the image will be written. Be **very careful** that the correct device is used as this command will destroy the existing data on the specified target device.

1. The command-line utility is available on BSD, Linux®, and Mac OS® systems. To burn the image using `dd`, insert the USB stick and determine its device name. Then, specify the name

of the downloaded installation file and the device name for the USB stick. This example burns the amd64 installation image to the first USB device on an existing FreeBSD system.

```
# dd if=FreeBSD-15.1-RELEASE-amd64-memstick.img of=/dev/da0 bs=1M conv=sync
```

If this command fails, verify that the USB stick is not mounted and that the device name is for the disk, not a partition.

Some operating systems might require this command to be run with sudo(8). The dd(1) syntax varies slightly across different platforms; for example, Mac OS® requires a lower-case `bs=1m`. Systems like Linux® might buffer writes. To force all writes to complete, use sync(8).

Procedure. Using Windows® to Write the Image

Be sure to give the correct drive letter as the existing data on the specified drive will be overwritten and destroyed.

1. **Obtaining Image Writer for Windows®**

   Image Writer for Windows® is a free application that can correctly write an image file to a memory stick. Download it from win32diskimager home page and extract it into a folder.

2. **Writing the Image with Image Writer**

   Double-click the Win32DiskImager icon to start the program. Verify that the drive letter shown under `Device` is the drive with the memory stick. Click the folder icon and select the image to be written to the memory stick. Click **[Save]** to accept the image file name. Verify that everything is correct, and that no folders on the memory stick are open in other windows. When everything is ready, click **[Write]** to write the image file to the memory stick.

2.4. Starting the Installation

By default, the installation will not make any changes to the disk(s) before the following message:

```
Your changes will now be written to disk. If you
have chosen to overwrite existing data, it will
be PERMANENTLY ERASED. Are you sure you want to
commit your changes?
```

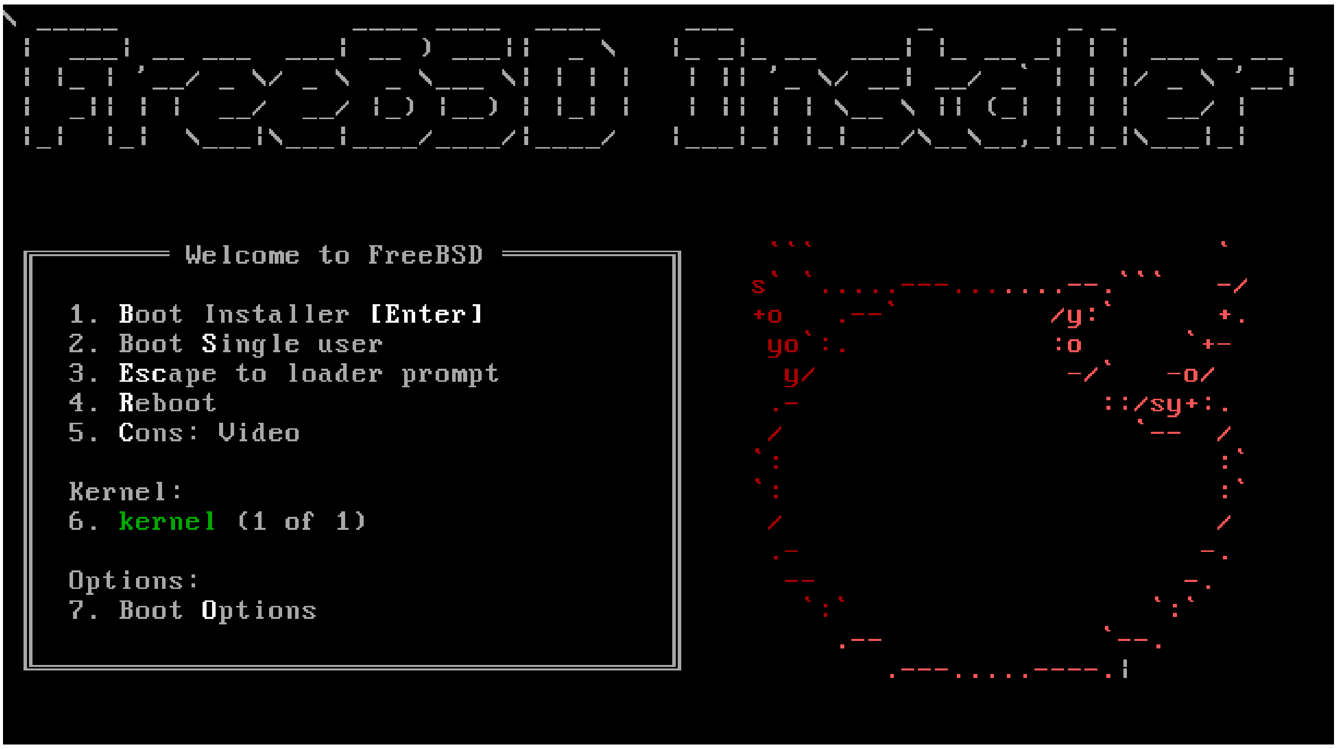
The install can be exited at any time prior to this warning. If there is a concern that something is incorrectly configured, just turn the computer off before this

> point and no changes will be made to the system's disks.

This section describes how to boot the system from the installation media which was prepared using the instructions in Prepare the Installation Media. When using a bootable USB stick, plug in the USB stick before turning on the computer. When booting from CD or DVD, turn on the computer and insert the media at the first opportunity. How to configure the system to boot from the inserted media depends upon the architecture.

2.4.1. FreeBSD Boot Loader Menu

Once the system boots from the installation media, a menu similar to the following will be displayed:



```
Welcome to FreeBSD

1. Boot Installer [Enter]
2. Boot Single user
3. Escape to loader prompt
4. Reboot
5. Cons: Video

Kernel:
6. kernel (1 of 1)

Options:
7. Boot Options
```

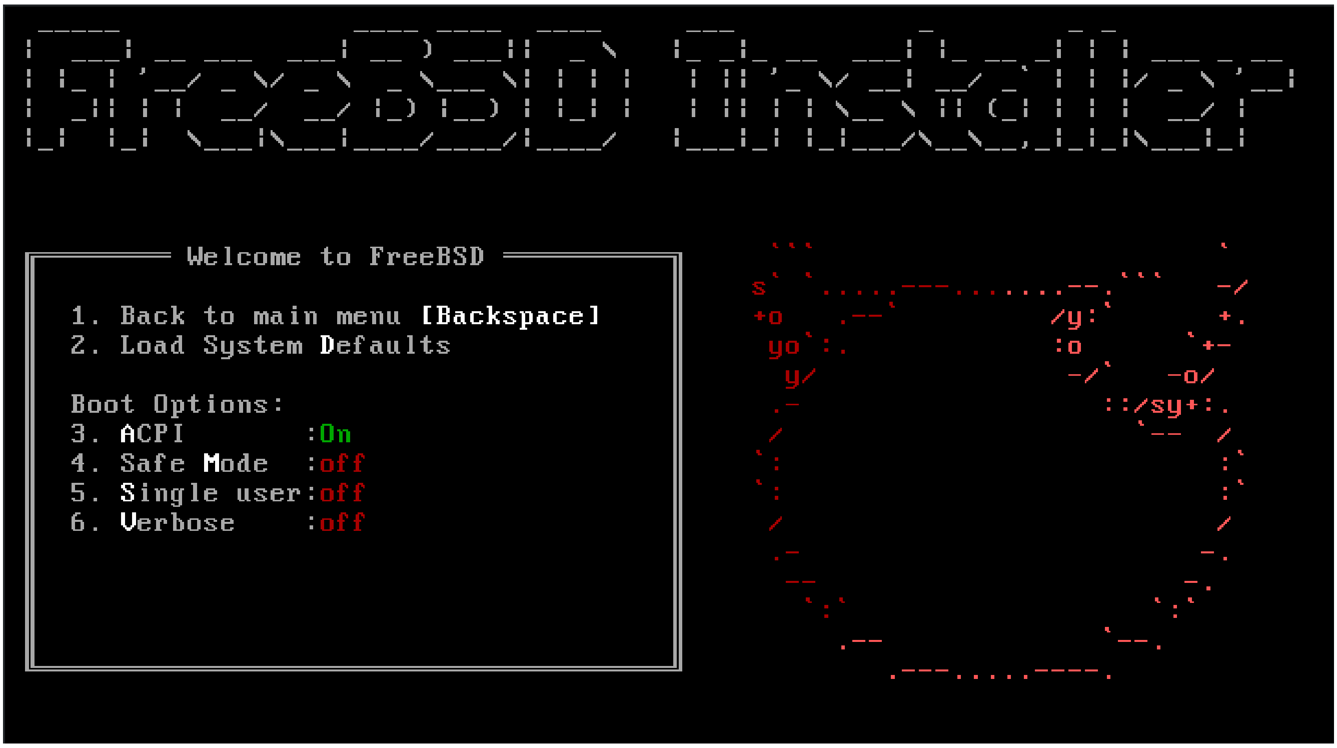
Figure 1. FreeBSD Boot Loader Menu

By default, the menu will wait ten seconds for user input before booting into the FreeBSD installer or, if FreeBSD is already installed, before booting into FreeBSD. To pause the boot timer in order to review the selections, press `Space`. To select an option, press its highlighted number, character, or key. The following options are available.

- `Boot Multi User`: This will continue the FreeBSD boot process. If the boot timer has been paused, press `1`, upper- or lower-case `B`, or `Enter`.
- `Boot Single User`: This mode can be used to fix an existing FreeBSD installation as described in "Single-User Mode". Press `2` or the upper- or lower-case `S` to enter this mode.
- `Escape to loader prompt`: This will boot the system into a repair prompt that contains a limited number of low-level commands. This prompt is described in "Stage Three". Press `3` or `Esc` to boot into this prompt.
- `Reboot`: Reboots the system.
- `Cons`: Allow to continue the installation by `video`, `serial`, `Dual (serial primary)` or `Dual (Video`

`primary)`

- `Kernel`: Loads a different kernel.
- `Boot Options`: Opens the menu shown in, and described under, FreeBSD Boot Options Menu.



```
Welcome to FreeBSD

1. Back to main menu [Backspace]
2. Load System Defaults

Boot Options:
3. ACPI        :On
4. Safe Mode   :off
5. Single user:off
6. Verbose     :off
```

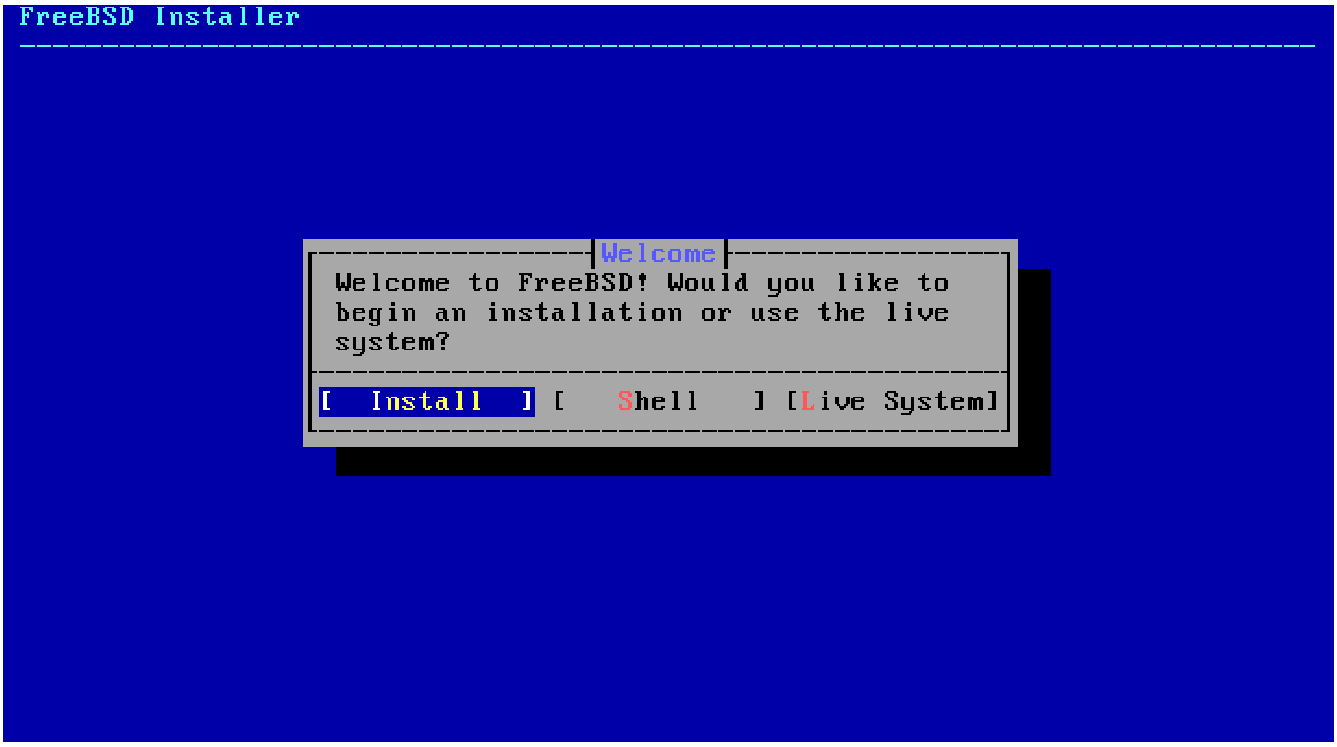
Figure 2. FreeBSD Boot Options Menu

The boot options menu is divided into two sections. The first section can be used to either return to the main boot menu or to reset any toggled options back to their defaults.

The next section is used to toggle the available options to `On` or `Off` by pressing the option's highlighted number or character. The system will always boot using the settings for these options until they are modified. Several options can be toggled using this menu:

- `ACPI`: If the system hangs during boot, try toggling this option to `Off`. This option is only present when ACPI support is available but not required.
- `Safe Mode`: If the system still hangs during boot even with `ACPI Support` set to `Off`, try setting this option to `On`.
- `Single User`: Toggle this option to `On` to fix an existing FreeBSD installation as described in "Single-User Mode". Once the problem is fixed, set it back to `Off`.
- `Verbose`: Toggle this option to `On` to see more detailed messages during the boot process. This can be useful when troubleshooting a piece of hardware.

After making the needed selections, press `1` or `Backspace` to return to the main boot menu, then press `Enter` to continue booting into FreeBSD. A series of boot messages will appear as FreeBSD carries out its hardware device probes and loads the installation program. Once the boot is complete, the welcome menu shown in Welcome Menu will be displayed.



Welcome to FreeBSD! Would you like to begin an installation or use the live system?

Figure 3. Welcome Menu

Press `Enter` to select the default of **[Install]** to enter the installer. The rest of this chapter describes how to use this installer. Otherwise, use the right or left arrows or the colorized letter to select the desired menu item. The **[Shell]** can be used to access a FreeBSD shell in order to use command line utilities to prepare the disks before installation. The **[Live CD]** option can be used to try out FreeBSD before installing it. The live version is described in Using the Live CD.

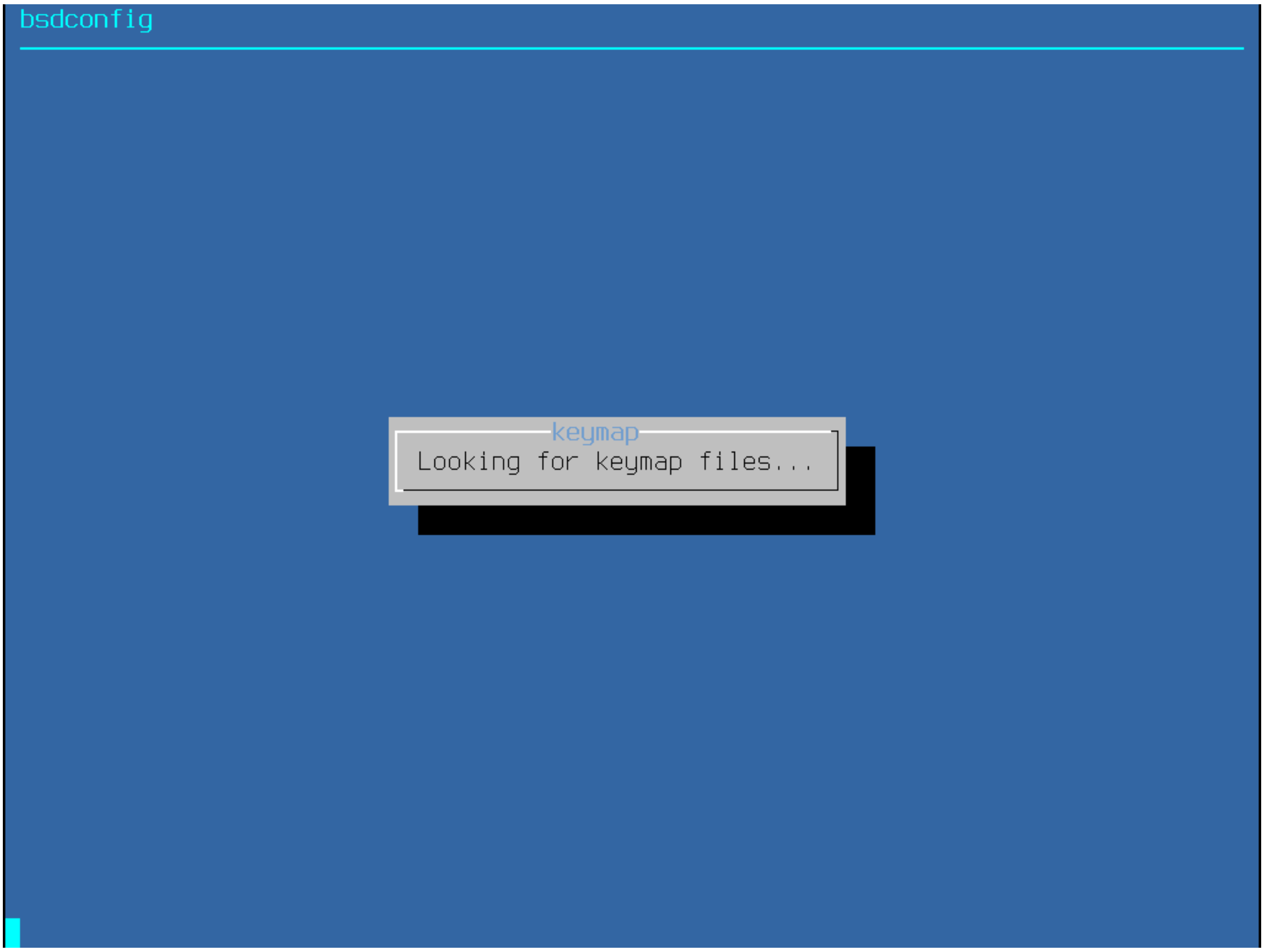
To review the boot messages, including the hardware device probe, press the upper- or lower-case `S` and then `Enter` to access a shell. At the shell prompt, type `more /var/run/dmesg.boot` and use the space bar to scroll through the messages. When finished, type `exit` to return to the welcome menu.

2.5. Using bsdinstall

This section shows the order of the bsdinstall menus and the type of information that will be asked before the system is installed. Use the arrow keys to highlight a menu option, then `Space` to select or deselect that menu item. When finished, press `Enter` to save the selection and move onto the next screen.

2.5.1. Selecting the Keymap Menu

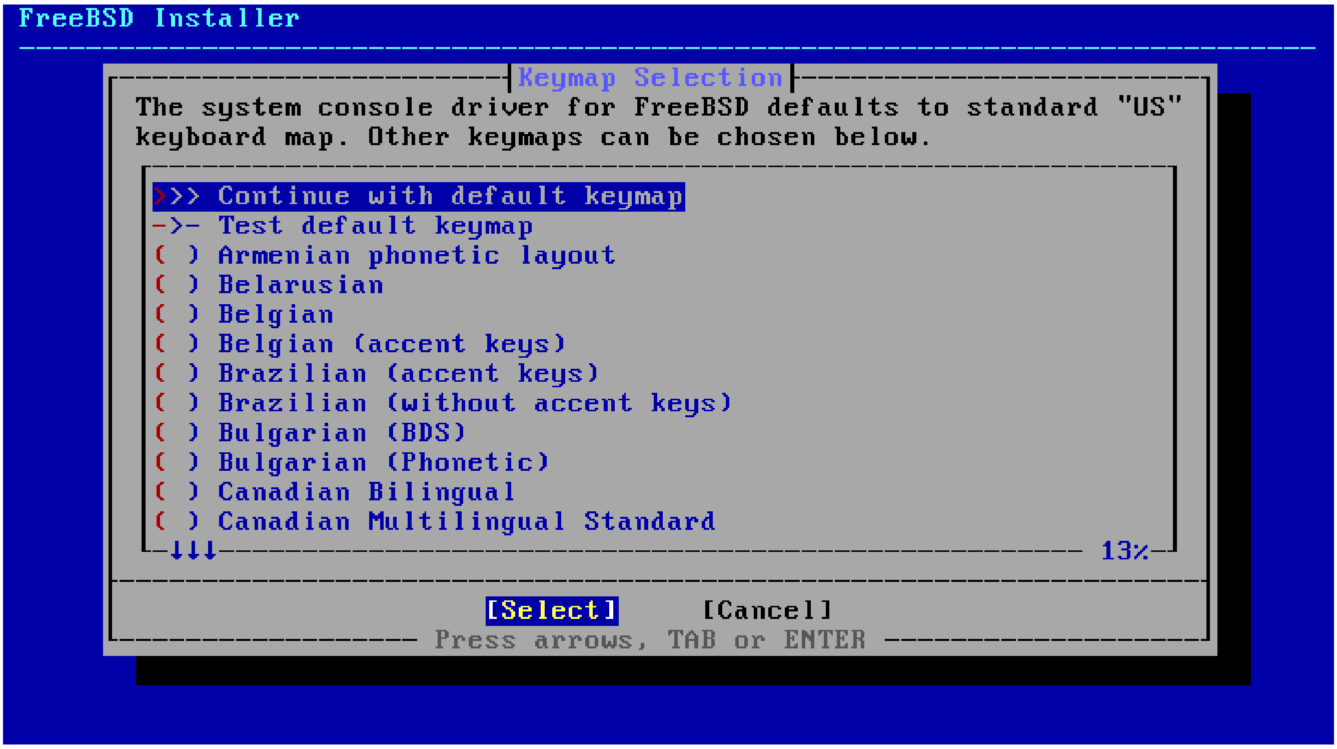
Before starting the process, bsdinstall will load the keymap files as shown in Keymap Loading.



```
Looking for keymap files...
```

Figure 4. Keymap Loading

After the keymaps have been loaded, bsdinstall displays the menu shown in Keymap Selection Menu. Use the up and down arrows to select the keymap that most closely represents the mapping of the keyboard attached to the system. Press `Enter` to save the selection.

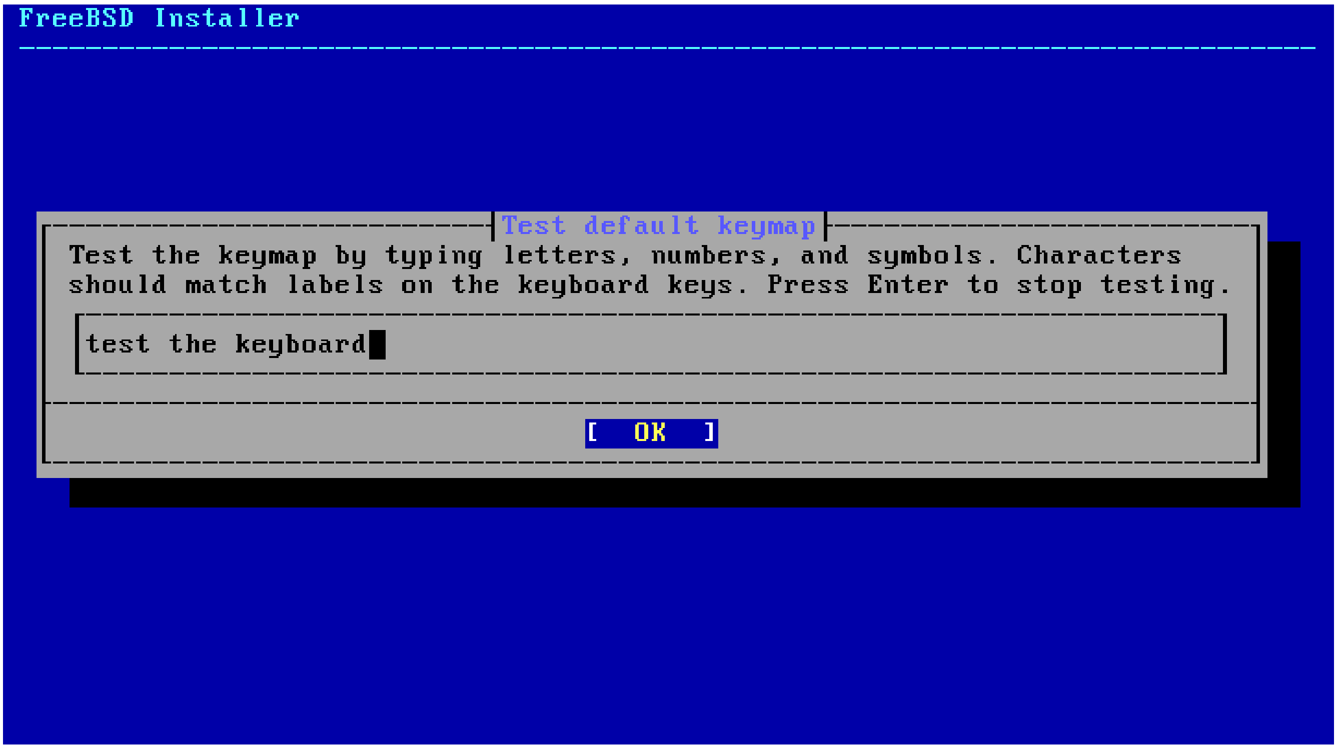

FreeBSD Installer
Keymap Selection
>>> Continue with default keymap
->- Test default keymap
( ) Armenian phonetic layout
( ) Belarusian
( ) Belgian
( ) Belgian (accent keys)
( ) Brazilian (accent keys)
( ) Brazilian (without accent keys)
( ) Bulgarian (BDS)
( ) Bulgarian (Phonetic)
( ) Canadian Bilingual
( ) Canadian Multilingual Standard
↓↓↓
13%
[Select]
[Cancel]
Press arrows, TAB or ENTER

The system console driver for FreeBSD defaults to standard "US" keyboard map. Other keymaps can be chosen below.

Figure 5. Keymap Selection Menu

Pressing `Esc` will exit this menu and use the default keymap. If the choice of keymap is not clear, United States of America ISO-8859-1 is also a safe option.

In addition, when selecting a different keymap, the user can try the keymap and ensure it is correct before proceeding, as shown in Keymap Testing Menu.


FreeBSD Installer
Test default keymap
[OK]

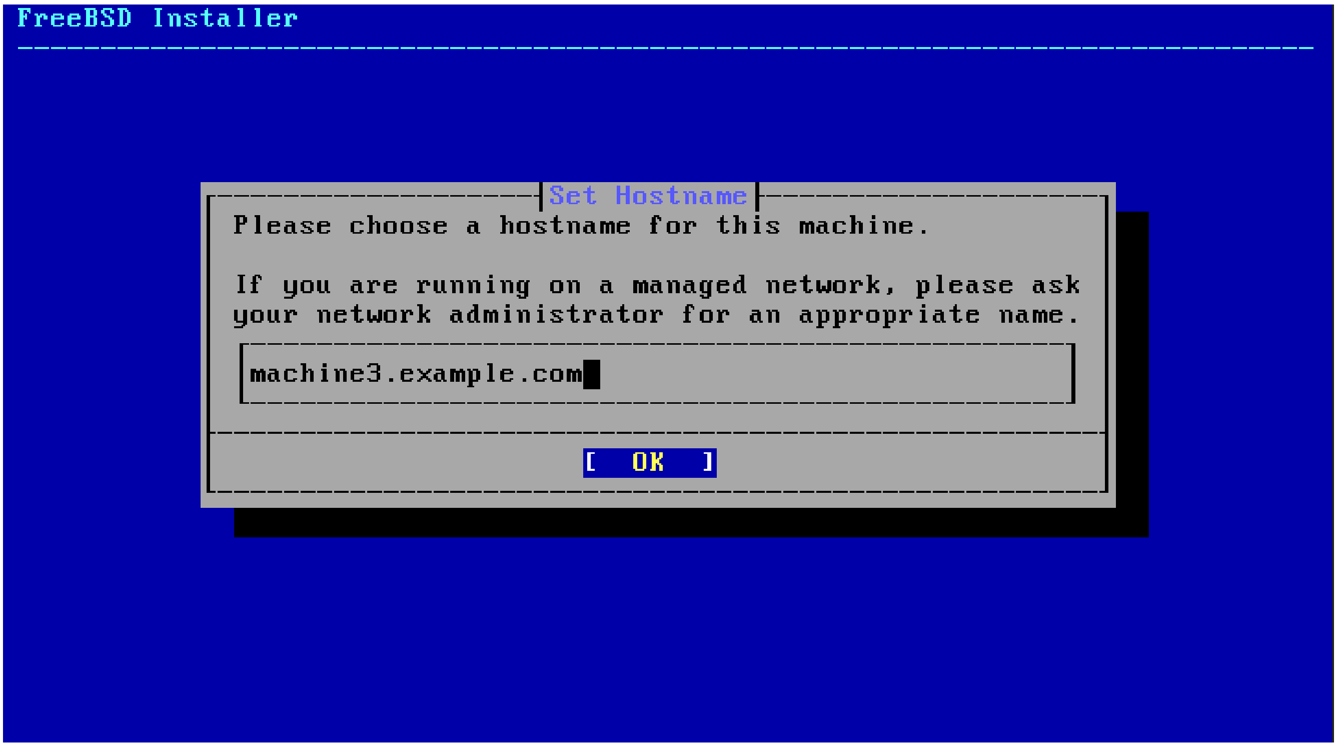
Test the keymap by typing letters, numbers, and symbols. Characters should match labels on the keyboard keys. Press Enter to stop testing.

test the keyboard

Figure 6. Keymap Testing Menu

2.5.2. Setting the Hostname

The next bsdinstall menu is used to set the hostname for the newly installed system.



Please choose a hostname for this machine.

If you are running on a managed network, please ask your network administrator for an appropriate name.

machine3.example.com

Figure 7. Setting the Hostname

Type in a hostname that is unique for the network. It should be a fully-qualified hostname, such as `machine3.example.com`.

2.5.3. Selecting Installation Type

In the next menu bsdinstall will ask whether to install the base system using traditional distribution sets or the new package-based method.

[Choice between "Distribution Sets" and "Packages (Tech Preview)"] | *bsdinstall-distributions-sets-pkgbase.png*

Figure 8. Selecting Installation Type

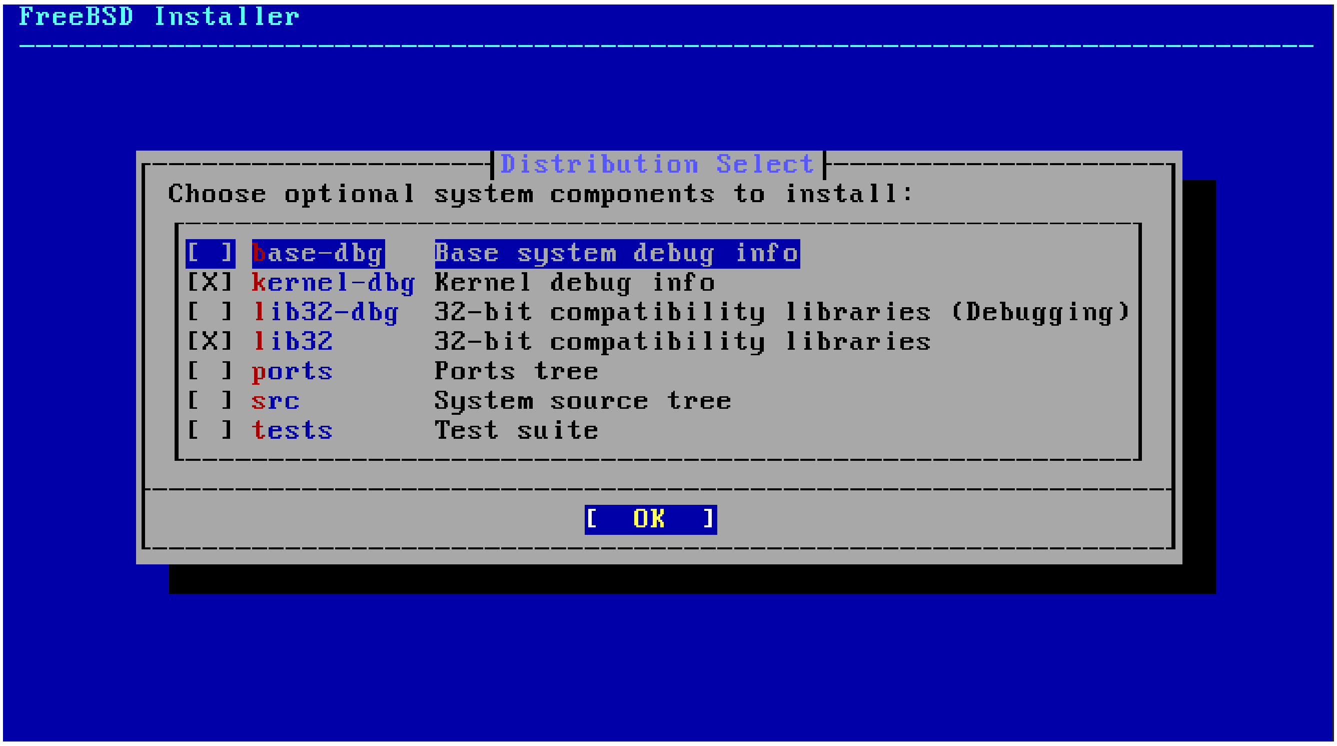
Starting with FreeBSD 15.0, the installer offers two distinct approaches for laying down and later updating the base system. Both options install the same kernel and userland but differ fundamentally in how they are delivered and maintained over time.

- `Distribution Sets` - The classic installation method used since the early days of the project. The base system is unpacked from a collection of compressed tarballs. Updates are performed exclusively with the freebsd-update(8) utility. This method is fully supported, stable, and remains the default for most installation media in the 15.x series.
- `Packages (Tech Preview)` - Installs the base system as a set of individual packages from the `FreeBSD-base` repository. Once installed, the entire operating system—both the base and any third-party software—can be upgraded with a single `pkg upgrade` command. This method, known as `pkgbase`, unifies system management under the pkg(8) framework.

The `Packages (Tech Preview)` option is offered for testing and early adoption. While it is expected to become the default in FreeBSD 16.0, users who prefer a conservative, time-tested update solution should select `Distribution Sets` for production environments during the 15.x lifecycle.

2.5.4. Selecting Components to Install

Next, bsdinstall will prompt to select optional components to install.



```
FreeBSD Installer
------------------------------------------------------------------------------

                        Distribution Select
   Choose optional system components to install:

   [ ] base-dbg    Base system debug info
   [X] kernel-dbg  Kernel debug info
   [ ] lib32-dbg   32-bit compatibility libraries (Debugging)
   [X] lib32       32-bit compatibility libraries
   [ ] ports       Ports tree
   [ ] src         System source tree
   [ ] tests       Test suite

                           [  OK  ]
```

Figure 9. Selecting Components to Install

Deciding which components to install will depend largely on the intended use of the system and the amount of disk space available. The FreeBSD kernel and userland, collectively known as the *base system*, are always installed. Depending on the architecture, some of these components may not appear:

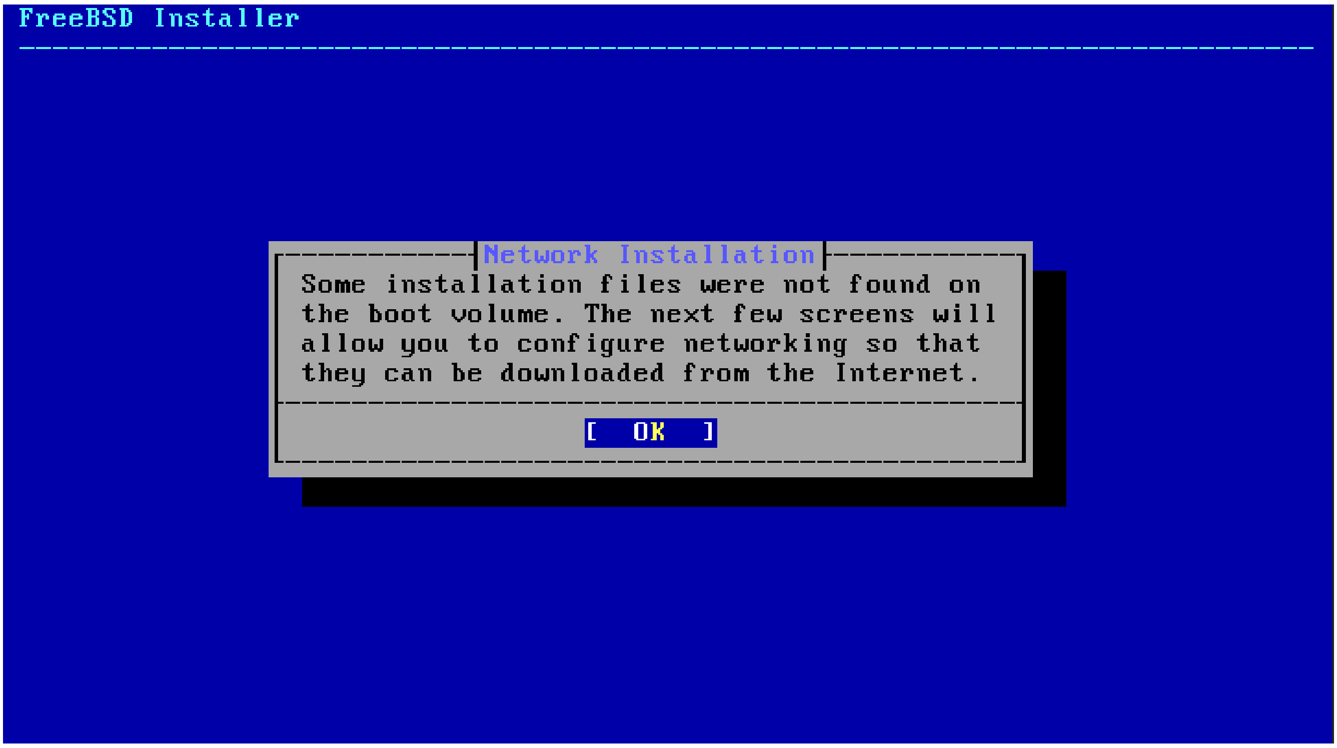
- `base-dbg` - Base tools like cat and ls, among many others, with debug symbols activated.
- `kernel-dbg` - Kernel and modules with debug symbols activated.
- `lib32-dbg` - Compatibility libraries for running 32-bit applications on a 64-bit version of FreeBSD with debug symbols activated.
- `lib32` - Compatibility libraries for running 32-bit applications on a 64-bit version of FreeBSD.
- `ports` - The FreeBSD Ports Collection is a collection of files which automates the downloading, compiling and installation of third-party software packages. Installing Applications: Packages and Ports discusses how to use the Ports Collection.

  The installation program does not check for adequate disk space. Select this option only if sufficient hard disk space is available. The FreeBSD Ports Collection takes up about 3 GB of disk space.

- `src` - The complete FreeBSD source code for both the kernel and the userland. Although not required for the majority of applications, it may be required to build device drivers, kernel modules, or some applications from the Ports Collection. It is also used for developing FreeBSD itself. The full source tree requires 1 GB of disk space and recompiling the entire FreeBSD system requires an additional 5 GB of space.
- `tests` - FreeBSD Test Suite.

2.5.5. Installing from the Network

The menu shown in Installing from the Network only appears when installing from a `-bootonly.iso` or `-mini-memstick.img`, as this installation media does not hold copies of the installation files. Since the installation files must be retrieved over a network connection, this menu indicates that the network interface must be configured first. If this menu is shown in any step of the process, remember to follow the instructions in Configuring Network Interfaces.

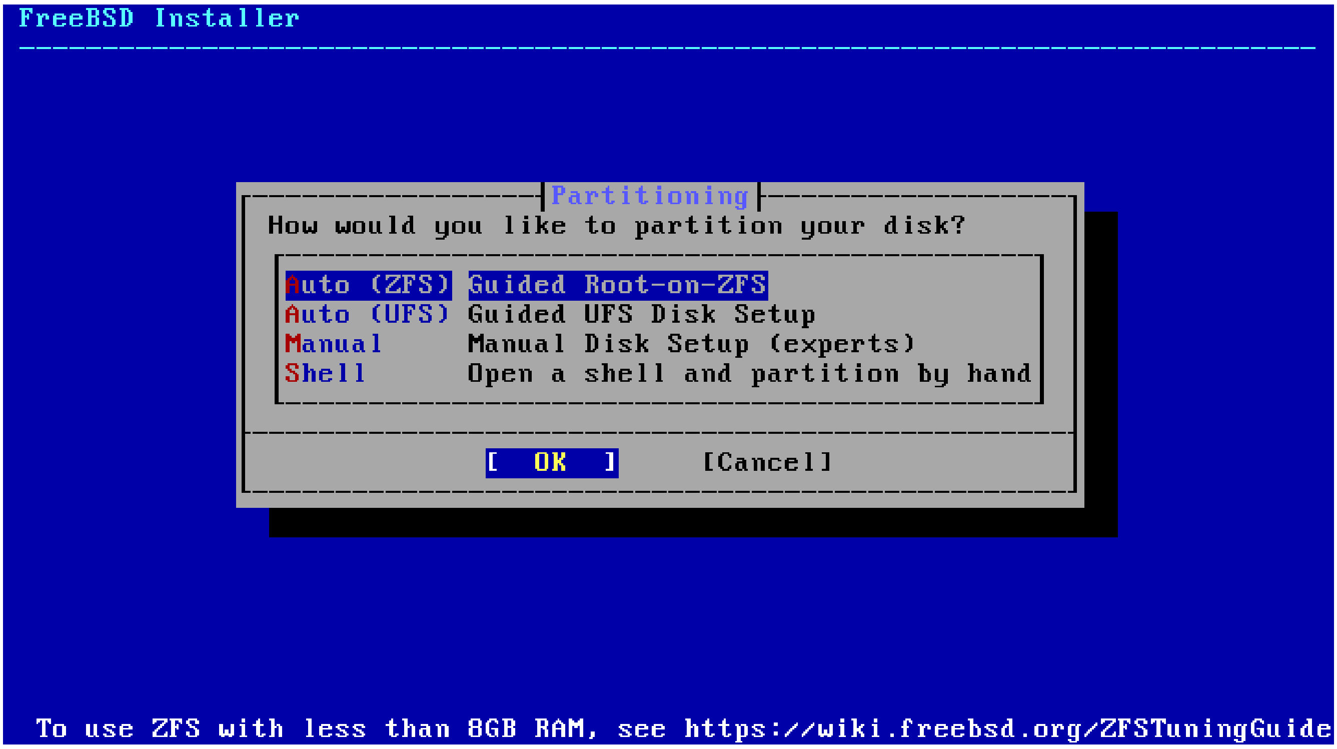


Some installation files were not found on the boot volume. The next few screens will allow you to configure networking so that they can be downloaded from the Internet.

Figure 10. Installing from the Network

2.6. Allocating Disk Space

The next menu is used to determine the method for allocating disk space.


FreeBSD Installer
Partitioning
Auto (ZFS) Guided Root-on-ZFS
Auto (UFS) Guided UFS Disk Setup
Manual Manual Disk Setup (experts)
Shell Open a shell and partition by hand
[OK]
[Cancel]

How would you like to partition your disk?

To use ZFS with less than 8GB RAM, see https://wiki.freebsd.org/ZFSTuningGuide

Figure 11. Partitioning Choices

bsdinstall gives the user four methods for allocating disk space:

- `Auto (ZFS)` partitioning creates a root-on-ZFS system with optional GELI encryption support for *boot environments*.
- `Auto (UFS)` partitioning automatically sets up the disk partitions using the `UFS` file system.
- `Manual` partitioning allows advanced users to create customized partitions from menu options.
- `Shell` opens a shell prompt where advanced users can create customized partitions using command-line utilities like gpart(8), fdisk(8), and bsdlabel(8).

This section describes what to consider when laying out the disk partitions. It then demonstrates how to use the different partitioning methods.

2.6.1. Designing the Partition Layout

The default partition layout for file systems includes one file system for the entire system. When using `UFS` it may be worth considering the use of multiple file systems with sufficient disk space or multiple disks available. When laying out file systems, remember that hard drives transfer data faster from the outer tracks to the inner. Thus, smaller and heavier-accessed file systems should be closer to the outside of the drive, while larger partitions like `/usr` should be placed toward the inner parts of the disk. It is a good idea to create partitions in an order similar to: `/`, swap, `/var`, and `/usr`.

The size of the `/var` partition reflects the intended machine's usage. This partition is used to hold mailboxes, log files, and printer spools. Mailboxes and log files can grow to unexpected sizes depending on the number of users and how long log files are kept. On average, most users rarely need more than about a gigabyte of free disk space in `/var`.

Sometimes, a lot of disk space is required in `/var/tmp`. When new software is installed, the packaging tools extract a temporary copy of the packages under

> `/var/tmp`. Large software packages, like Firefox or LibreOffice may be tricky to install if there is not enough disk space under `/var/tmp`.

The `/usr` partition holds many of the files which support the system, including the FreeBSD Ports Collection and system source code. At least 2 gigabytes of space is recommended for this partition. Also, note that home directories for users are placed in `/usr/home` by default, but can be placed on another partition. By default, `/home` is a symbolic link to `/usr/home`.

When selecting partition sizes, keep the space requirements in mind. Running out of space in one partition while barely using another can be a hassle.

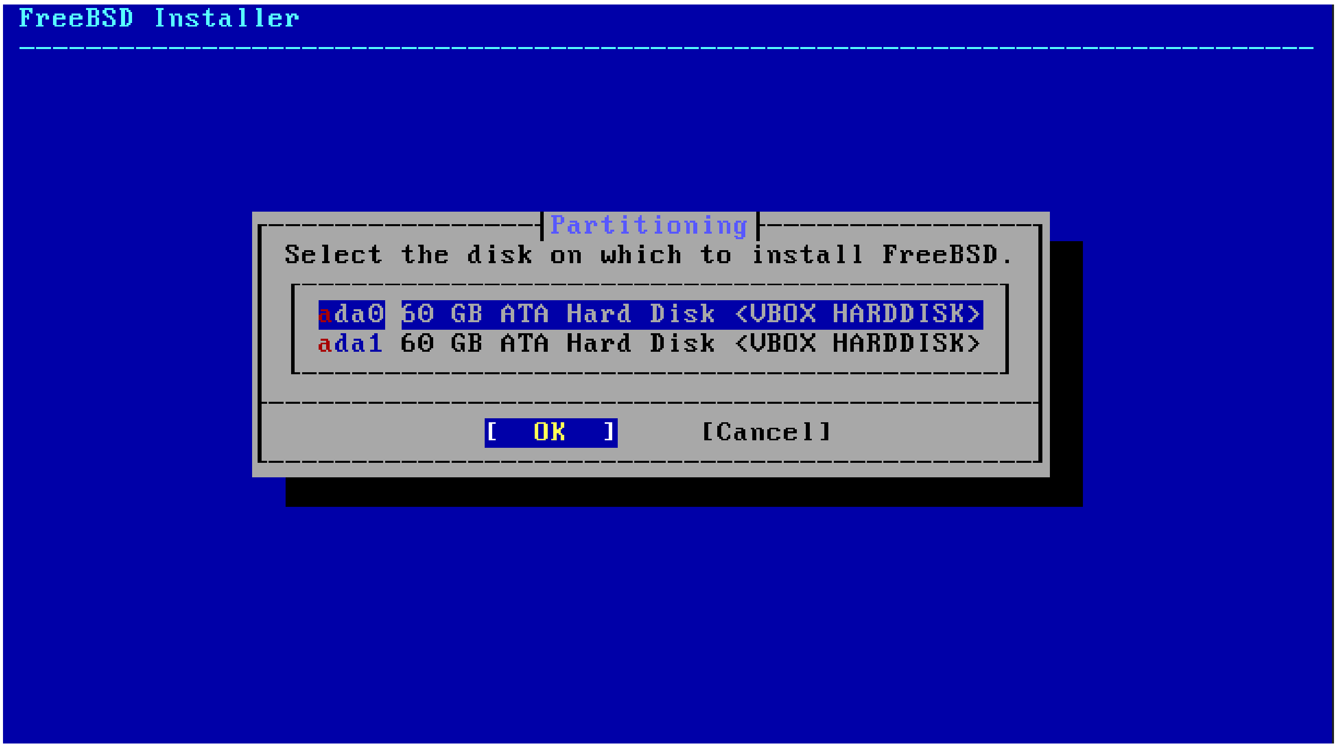
As a rule of thumb, the swap partition should be about double the size of physical memory (RAM). Systems with minimal RAM (less for larger-memory configurations) may perform better with more swap. Configuring too little swap can lead to inefficiencies in the VM page scanning code and might create issues later if more memory is added.

On larger systems with multiple SCSI disks or multiple IDE disks operating on different controllers, it is recommended that swap be configured on each drive, up to four drives. The swap partitions should be approximately the same size. The kernel can handle arbitrary sizes, but internal data structures scale to 4 times the largest swap partition. Keeping the swap partitions near the same size will allow the kernel to optimally stripe swap space across disks. Large swap sizes may elicit a kernel warning message about the total configured swap. The limit is raised by increasing the amount of memory allowed for keeping track of swap allocations, as instructed by the warning message. It might be easier to recover from a runaway program before being forced to reboot.

By properly partitioning a system, fragmentation introduced in the smaller write-heavy partitions will not bleed over into the mostly read partitions. Keeping the write-loaded partitions closer to the disk's edge will increase I/O performance in the partitions where it occurs the most. While I/O performance in the larger partitions may be needed, shifting them more toward the edge of the disk will not lead to a significant performance improvement over moving `/var` to the edge.

2.6.2. Guided Partitioning Using UFS

When this method is selected, a menu will display the available disk(s). If multiple disks are connected, choose the one where FreeBSD is to be installed.



FreeBSD Installer

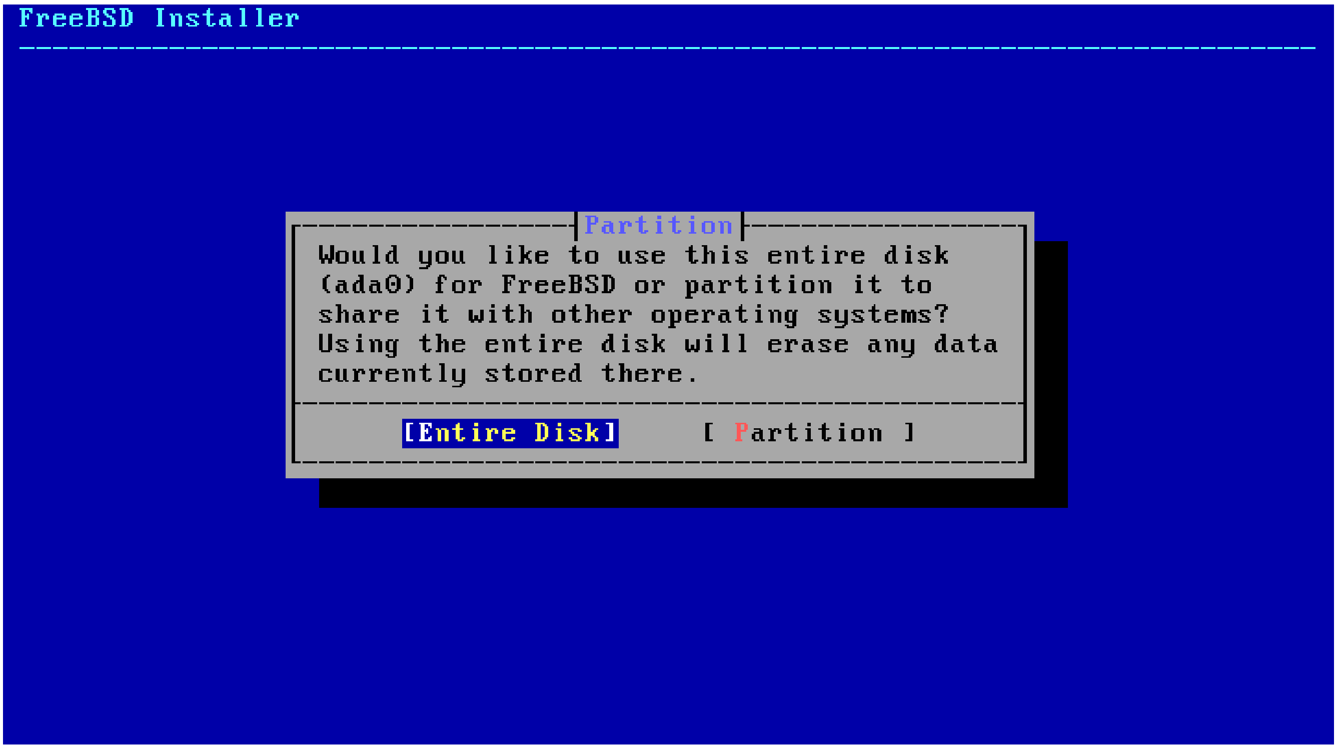
Partitioning

Select the disk on which to install FreeBSD.

ada0 60 GB ATA Hard Disk <VBOX HARDDISK>
ada1 60 GB ATA Hard Disk <VBOX HARDDISK>
[OK]
[Cancel]

Figure 12. Selecting from Multiple Disks

Once the disk is selected, the next menu prompts to install to either the entire disk or to create a partition using free space. If **[Entire Disk]** is chosen, a general partition layout filling the whole disk is automatically created. Selecting **[Partition]** creates a partition layout from the unused space on the disk.



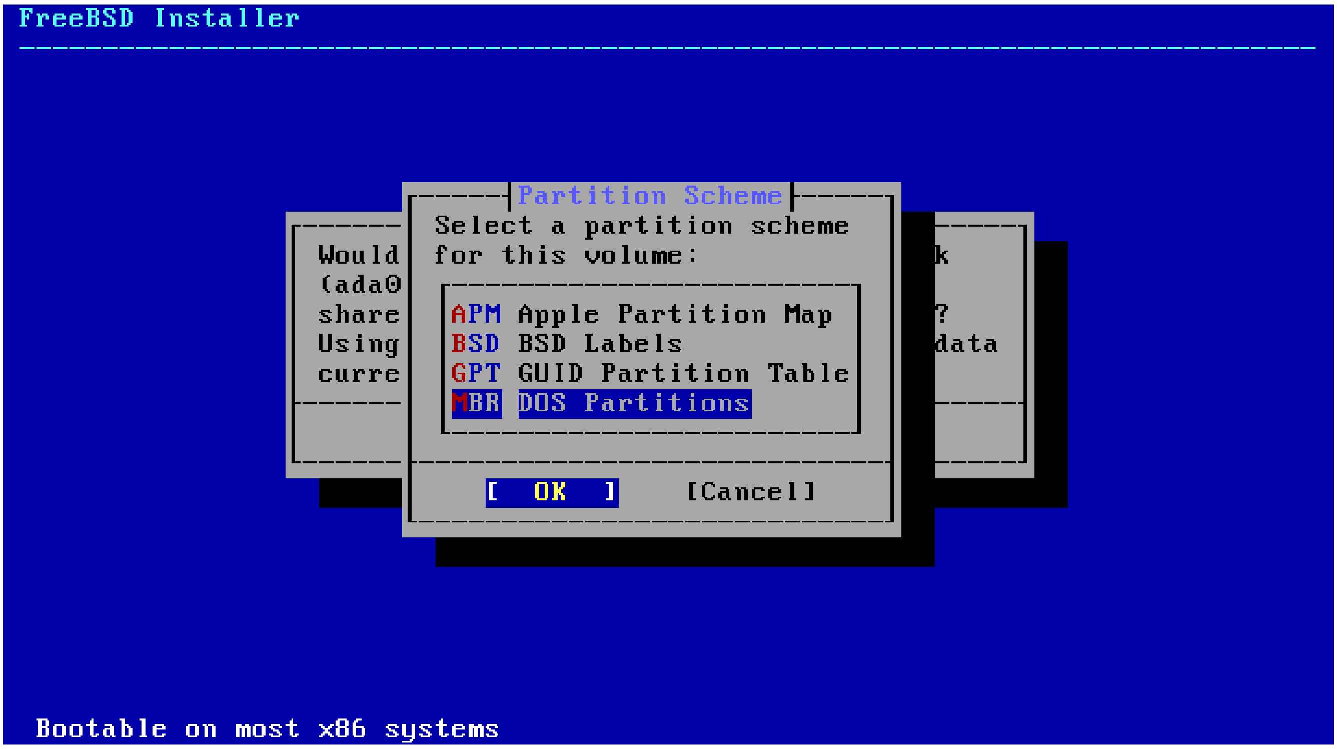
FreeBSD Installer

Partition

Would you like to use this entire disk (ada0) for FreeBSD or partition it to share it with other operating systems? Using the entire disk will erase any data currently stored there.

Figure 13. Selecting Entire Disk or Partition

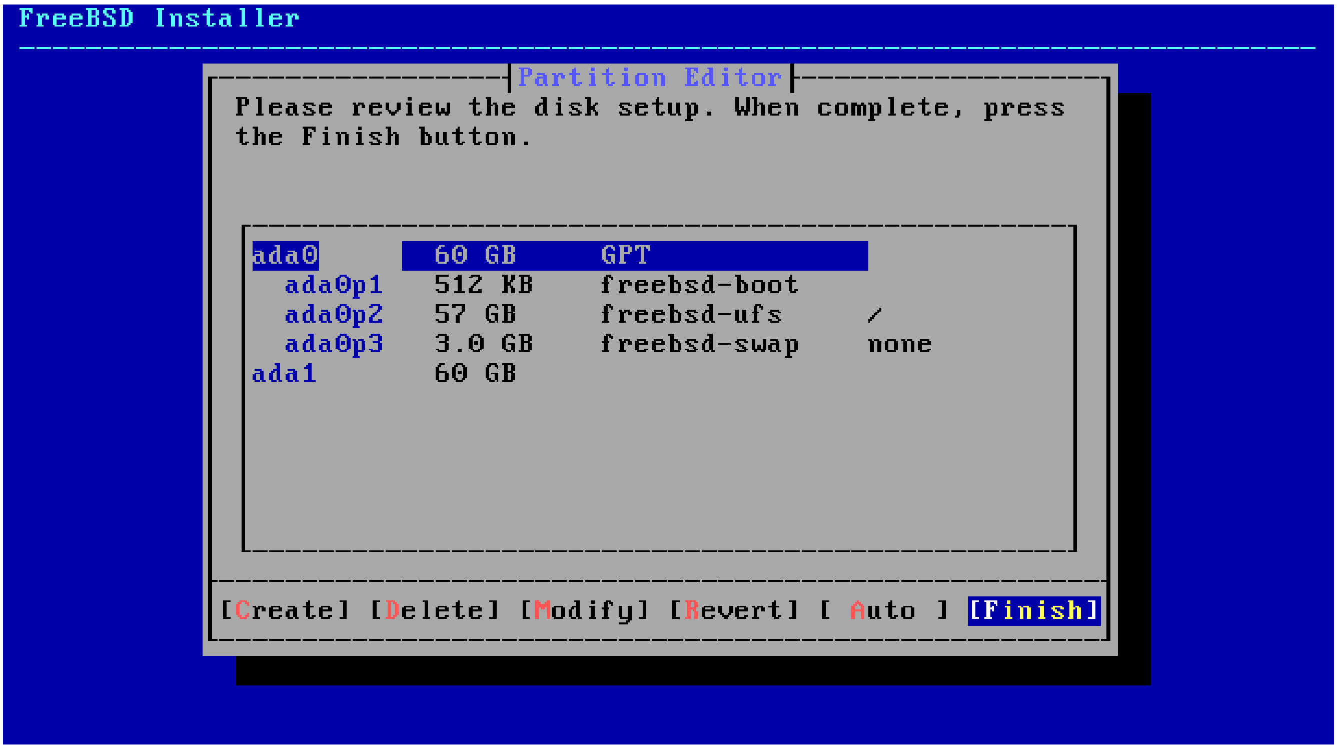
The next menu shows a list with the available partition scheme types. GPT is usually the most appropriate choice for amd64 computers. Older computers that are not compatible with GPT should use MBR. The other partition schemes are generally used for uncommon or older computers. More information is available in Partitioning Schemes.


FreeBSD Installer
Partition Scheme
APM Apple Partition Map
BSD BSD Labels
GPT GUID Partition Table
MBR DOS Partitions
[OK]
[Cancel]

```
Select a partition scheme
for this volume:

Would
(ada0
share
Using
curre

k

?
data

Bootable on most x86 systems
```

Figure 14. Select Partition Scheme

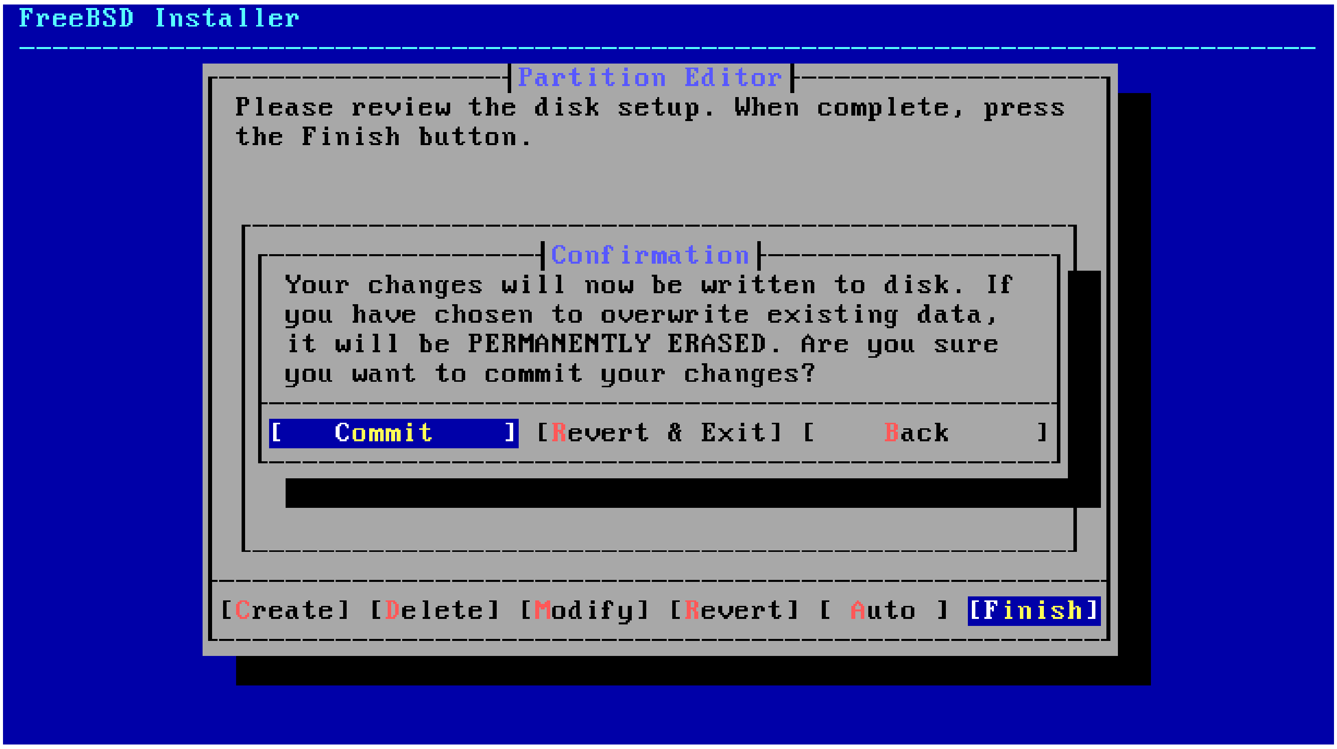
After the partition layout has been created, review it to ensure it meets the needs of the installation. Selecting **[Revert]** will reset the partitions to their original values. Pressing **[Auto]** will recreate the automatic FreeBSD partitions. Partitions can also be manually created, modified, or deleted. When the partitioning is correct, select **[Finish]** to continue with the installation.


FreeBSD Installer
Partition Editor
[Create] [Delete] [Modify] [Revert] [Auto] [Finish]

```
Please review the disk setup. When complete, press
the Finish button.

ada0        60 GB       GPT
  ada0p1    512 KB      freebsd-boot
  ada0p2    57 GB       freebsd-ufs     /
  ada0p3    3.0 GB      freebsd-swap    none
ada1        60 GB
```

Figure 15. Review Created Partitions

Once the disks are configured, the next menu provides the last chance to make changes before the selected drives are formatted. If changes need to be made, select **[Back]** to return to the main partitioning menu. **[Revert & Exit]** exits the installer without making any changes to the drive. Otherwise, select **[Commit]** to start the installation process.


FreeBSD Installer

Partition Editor

Please review the disk setup. When complete, press the Finish button.

Confirmation

Your changes will now be written to disk. If you have chosen to overwrite existing data, it will be PERMANENTLY ERASED. Are you sure you want to commit your changes?

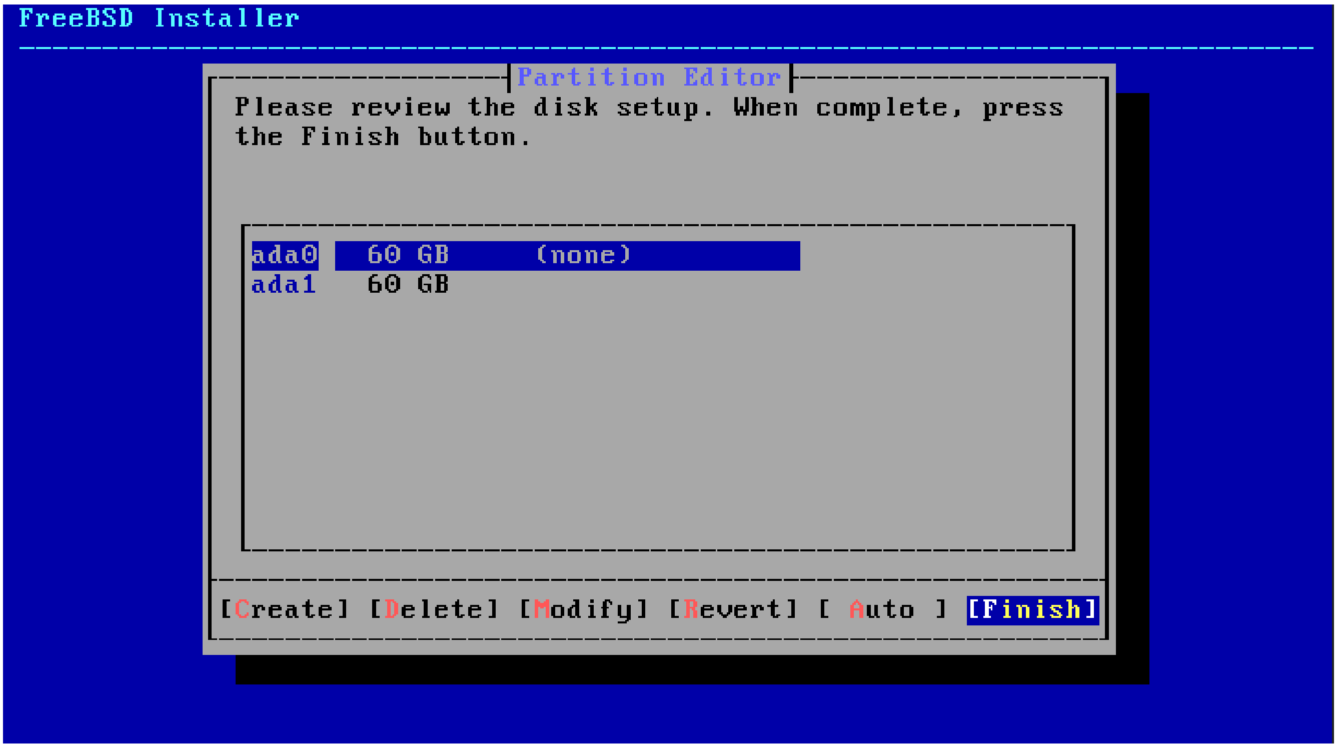
[Commit]
[Revert & Exit]
[Back]
[Create]
[Delete]
[Modify]
[Revert]
[Auto]
[Finish]

Figure 16. Final Confirmation

To continue with the installation process, go to Fetching Distribution Files.

2.6.3. Manual Partitioning

Selecting this method opens the partition editor:


FreeBSD Installer

Partition Editor

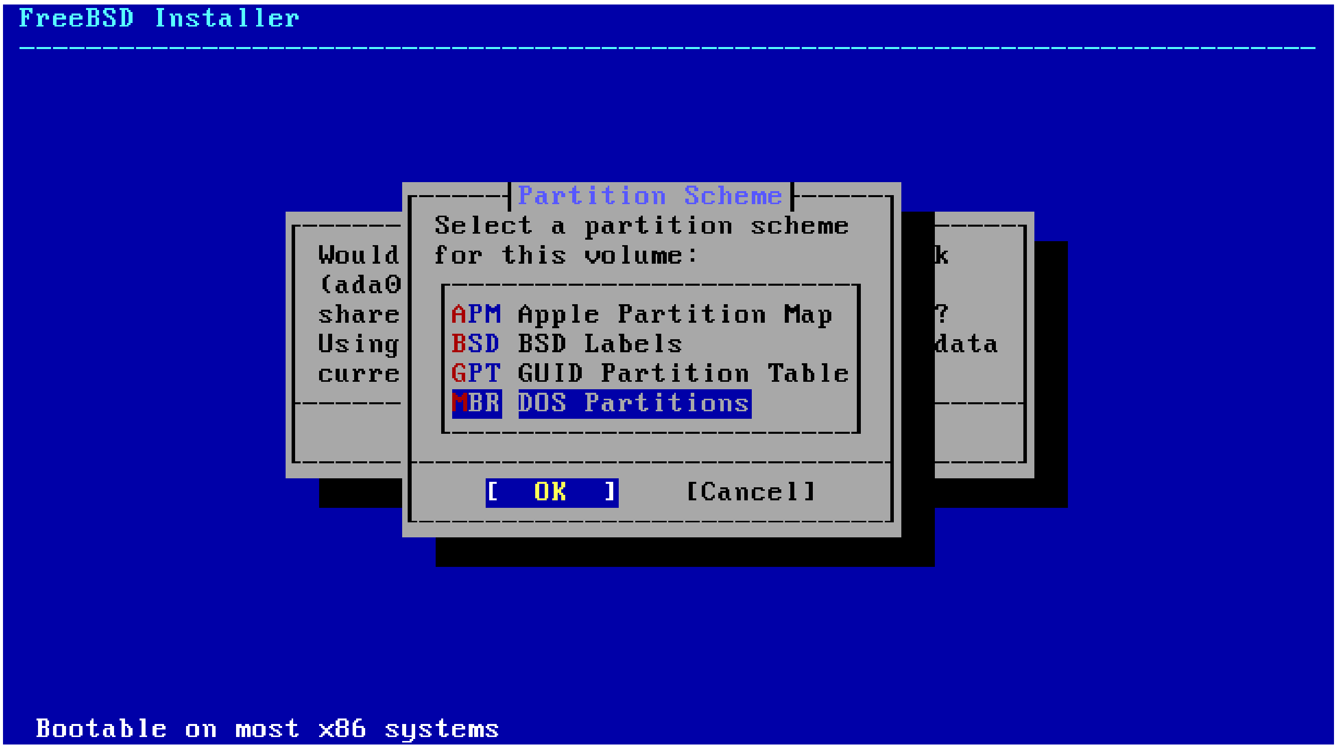
Please review the disk setup. When complete, press the Finish button.

```
ada0    60 GB     (none)
ada1    60 GB
```

[Create]
[Delete]
[Modify]
[Revert]
[Auto]
[Finish]

Figure 17. Manually Create Partitions

Highlight the installation drive (`ada0` in this example) and select **[Create]** to display a menu of available partition schemes:



FreeBSD Installer

Partition Scheme

Select a partition scheme for this volume:

APM Apple Partition Map
BSD BSD Labels
GPT GUID Partition Table
MBR DOS Partitions
[OK]
[Cancel]
Would
(ada0
share
Using
curre
k
?
data

Bootable on most x86 systems

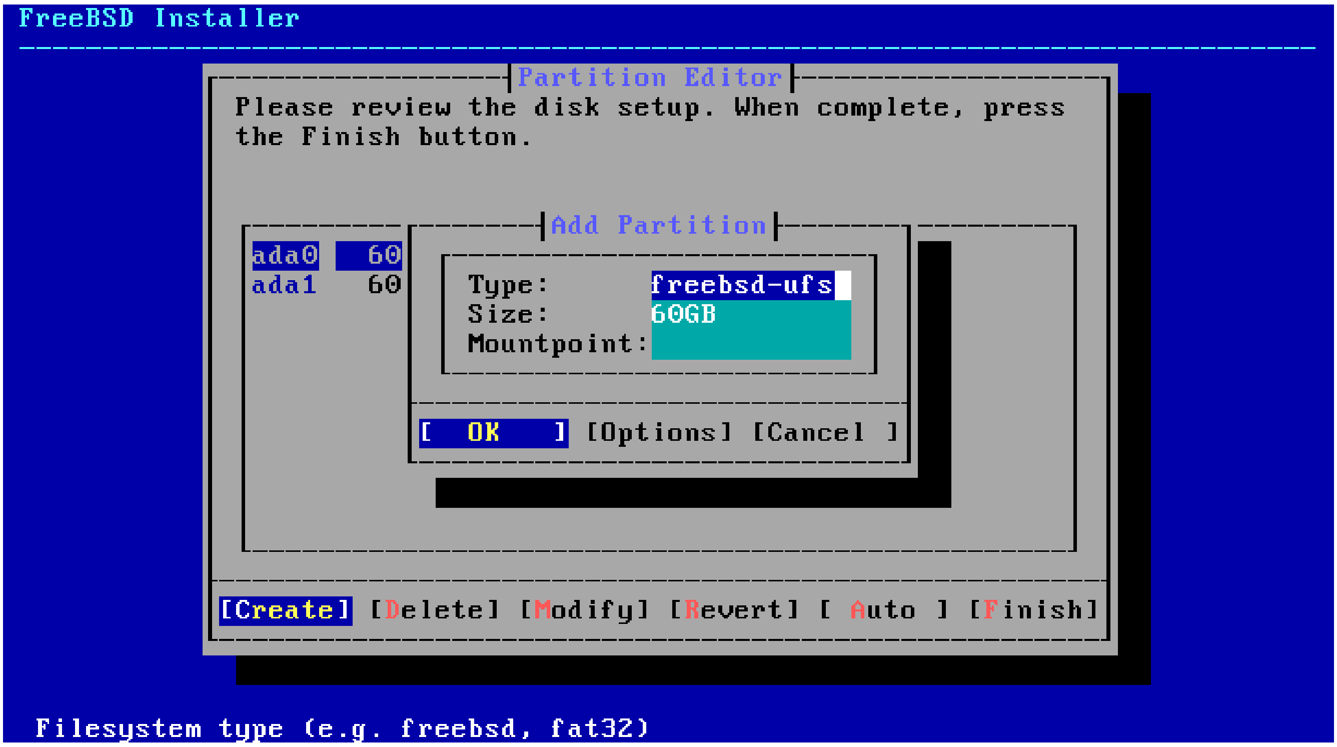
Figure 18. Manually Create Partitions

GPT is usually the most appropriate choice for amd64 computers. Older computers that are not compatible with GPT should use MBR. The other partition schemes are generally used for uncommon or older computers.

Table 1. Partitioning Schemes

Abbreviation	Description
APM	Apple Partition Map, used by PowerPC®.
BSD	BSD label without an MBR, sometimes called *dangerously dedicated mode* as non-BSD disk utilities may not recognize it.
GPT	GUID Partition Table.
MBR	Master Boot Record.

After the partitioning scheme has been selected and created, select **[Create]** again to create the partitions. The `Tab` key is used to give focus to the fields (after cycling through **[<OK>]**, **[<Options>]**, and **[<Cancel>]**).



Please review the disk setup. When complete, press the Finish button.

```
ada0   60
ada1   60
```

```
Type:        freebsd-ufs
Size:        60GB
Mountpoint:
```

Filesystem type (e.g. freebsd, fat32)

Figure 19. Manually Create Partitions

A standard FreeBSD GPT installation uses at least three partitions, including either UFS or ZFS:

- `freebsd-boot` or `efi` - Holds the FreeBSD boot code.
- `freebsd-ufs` - A FreeBSD UFS file system.
- `freebsd-zfs` - A FreeBSD ZFS file system. More information about ZFS is available in The Z File System (ZFS).
- `freebsd-swap` - FreeBSD swap space.

Refer to gpart(8) for descriptions of the available GPT partition types.

Multiple file system partitions can be created. Some people prefer a traditional layout with separate partitions for `/`, `/var`, `/tmp`, and `/usr`.

> Note that `/tmp` can be added later as a memory-based file system (tmpfs(5)) on systems with sufficient memory.

See Creating Traditional Split File System Partitions for an example.

The `Size` may be entered with common abbreviations: *K* for kilobytes, *M* for megabytes, or *G* for gigabytes.

> Proper sector alignment provides the best performance, and making partition sizes even multiples of 4K bytes helps to ensure alignment on drives with either 512-byte or 4K-byte sectors. Generally, using partition sizes that are even multiples of 1M or 1G is the easiest way to make sure every partition starts at an even multiple of 4K. There is one exception: a *freebsd-boot* partition for BIOS booting should be no larger than 512K due to legacy boot code limitations. This limitation does not apply to UEFI booting.

A `Mountpoint` is needed if the partition will contain a file system. If only a single UFS partition will be created, the mountpoint should be `/`.

The `Label` is a name by which the partition will be known. Drive names or numbers can change if the drive is connected to a different controller or port, but the partition label does not change. Referring to labels instead of drive names and partition numbers in files like `/etc/fstab` makes the system more tolerant to hardware changes. GPT labels appear in `/dev/gpt/` when a disk is attached. Other partitioning schemes have different label capabilities and their labels appear in different directories in `/dev/`.

Use a unique label on every partition to avoid conflicts from identical labels. A few letters from the computer's name, use, or location can be added to the label. For instance, use `labroot` or `rootfslab` for the UFS root partition on the computer named `lab`.

Example 1. Creating Traditional Split File System Partitions

For a traditional partition layout where the `/`, `/var`, `/tmp`, and `/usr` directories are separate file systems on their own partitions, create a GPT partitioning scheme, then create the partitions as shown. Partition sizes shown are typical for a 20G target disk. If more space is available on the target disk, larger swap or `/var` partitions may be useful. Labels shown here are prefixed with `ex` for "example", but readers should use other unique label values as described above.

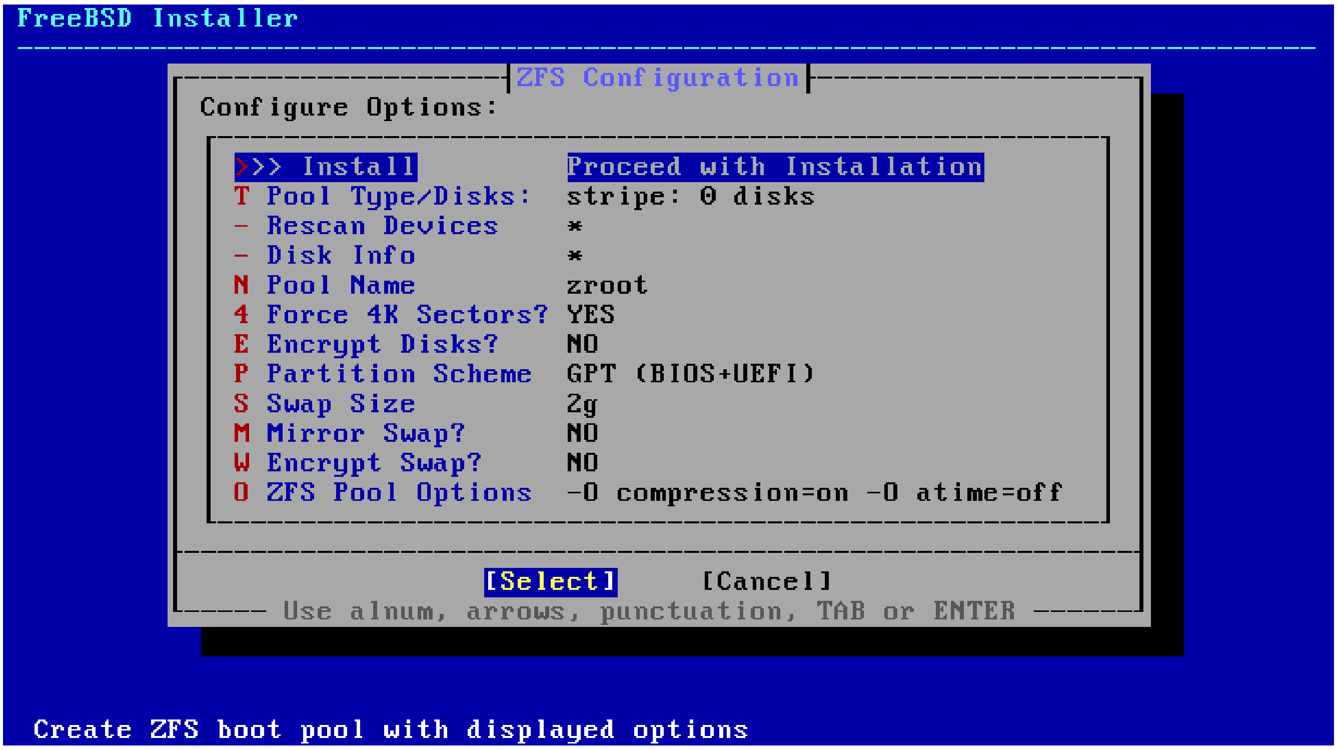
By default, FreeBSD's `gptboot` expects the first UFS partition to be the `/` partition.

Partition Type	Size	Mountpoint	Label
`freebsd-boot`	`512K`		
`freebsd-ufs`	`2G`	`/`	`exrootfs`
`freebsd-swap`	`4G`		`exswap`
`freebsd-ufs`	`2G`	`/var`	`exvarfs`
`freebsd-ufs`	`1G`	`/tmp`	`extmpfs`
`freebsd-ufs`	accept the default (remainder of the disk)	`/usr`	`exusrfs`

After the custom partitions have been created, select **[Finish]** to continue with the installation and go to Fetching Distribution Files.

2.6.4. Guided Partitioning Using Root-on-ZFS

This partitioning mode only works with whole disks and will erase the contents of the entire disk. The main ZFS configuration menu offers a number of options to control the creation of the pool.



```
FreeBSD Installer
                          ZFS Configuration
  Configure Options:

     >>> Install            Proceed with Installation
     T Pool Type/Disks:     stripe: 0 disks
     - Rescan Devices       *
     - Disk Info            *
     N Pool Name            zroot
     4 Force 4K Sectors?    YES
     E Encrypt Disks?       NO
     P Partition Scheme     GPT (BIOS+UEFI)
     S Swap Size            2g
     M Mirror Swap?         NO
     W Encrypt Swap?        NO
     O ZFS Pool Options     -O compression=on -O atime=off

                      [Select]      [Cancel]
        Use alnum, arrows, punctuation, TAB or ENTER

 Create ZFS boot pool with displayed options
```

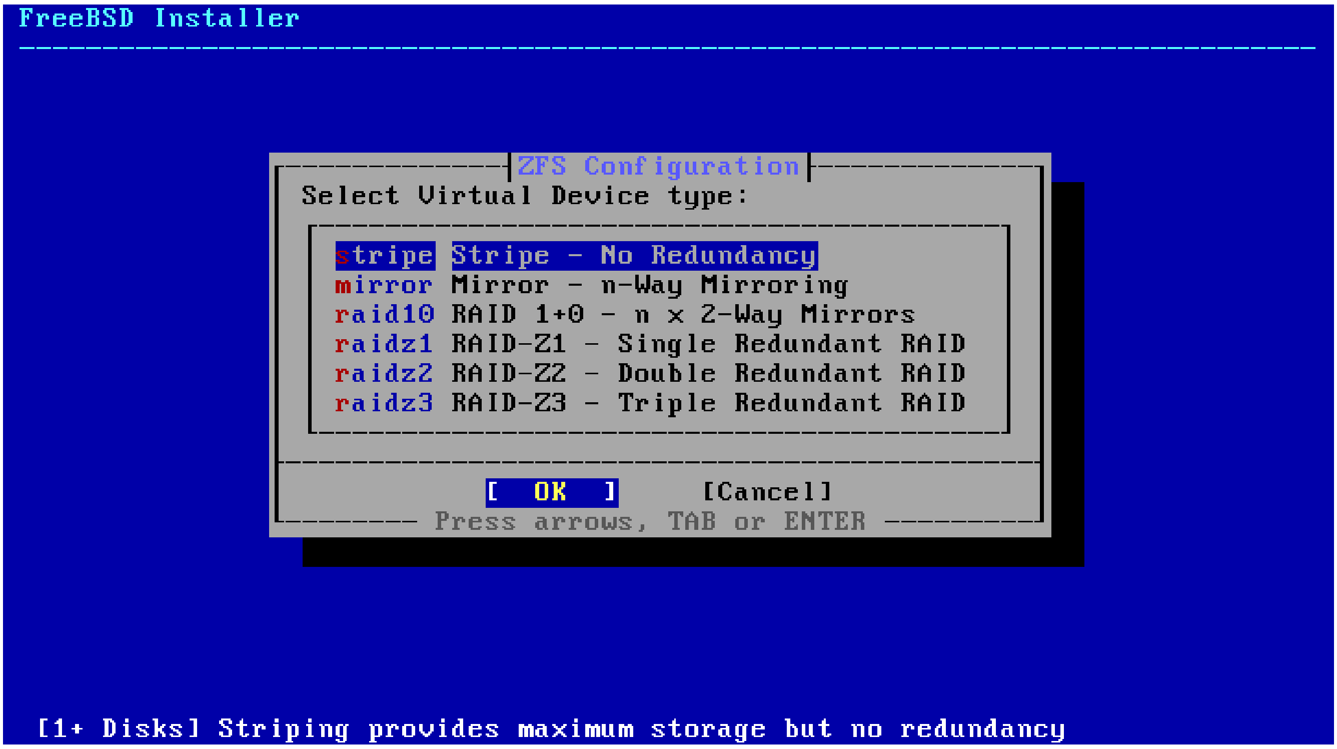
Figure 20. ZFS Partitioning Menu

Here is a summary of the options in this menu:

- `Install` - Proceed with the installation with the selected options.
- `Pool Type/Disks` - Configure the `Pool Type` and the disk(s) that will constitute the pool. The automatic ZFS installer currently only supports the creation of a single top level vdev, except in stripe mode. To create more complex pools, use the instructions in Shell Mode Partitioning to create the pool.
- `Rescan Devices` - Repopulate the list of available disks.
- `Disk Info` - This menu can be used to inspect each disk, including its partition table and various other information such as the device model number and serial number, if available.
- `Pool Name` - Establish the name of the pool. The default name is *zroot*.
- `Force 4K Sectors?` - Force the use of 4K sectors. By default, the installer will automatically create partitions aligned to 4K boundaries and force ZFS to use 4K sectors. This is safe even with 512 byte sector disks, and has the added benefit of ensuring that pools created on 512 byte disks will be able to have 4K sector disks added in the future, either as additional storage space or as replacements for failed disks. Press the `Enter` key to chose to activate it or not.
- `Encrypt Disks?` - Encrypting the disks allows the user to encrypt the disks using GELI. More information about disk encryption is available in "Disk Encryption with geli". Press the `Enter` key to choose whether to activate it or not.
- `Partition Scheme` - Choose the partition scheme. GPT is the recommended option in most cases. Press the `Enter` key to chose between the different options.
- `Swap Size` - Establish the amount of swap space.
- `Mirror Swap?` - Whether to mirror the swap between the disks. Be aware that enabling mirror swap will break crash dumps. Press the `Enter` key to activate it or not.

- `Encrypt Swap?` - Whether to encrypt the swap. This will encrypt the swap with a temporary key each time the system boots, and discards it on reboot. Press the `Enter` key to choose to activate it or not. More information about swap encryption in "Encrypting Swap".
- `ZFS Pool Options` - Specifies additional pool properties to apply when the ZFS root pool is created. More information about the options can be found in zpool-create(8) and zpoolprops(7).

Select `T` to configure the `Pool Type` and the disk(s) that will constitute the pool.



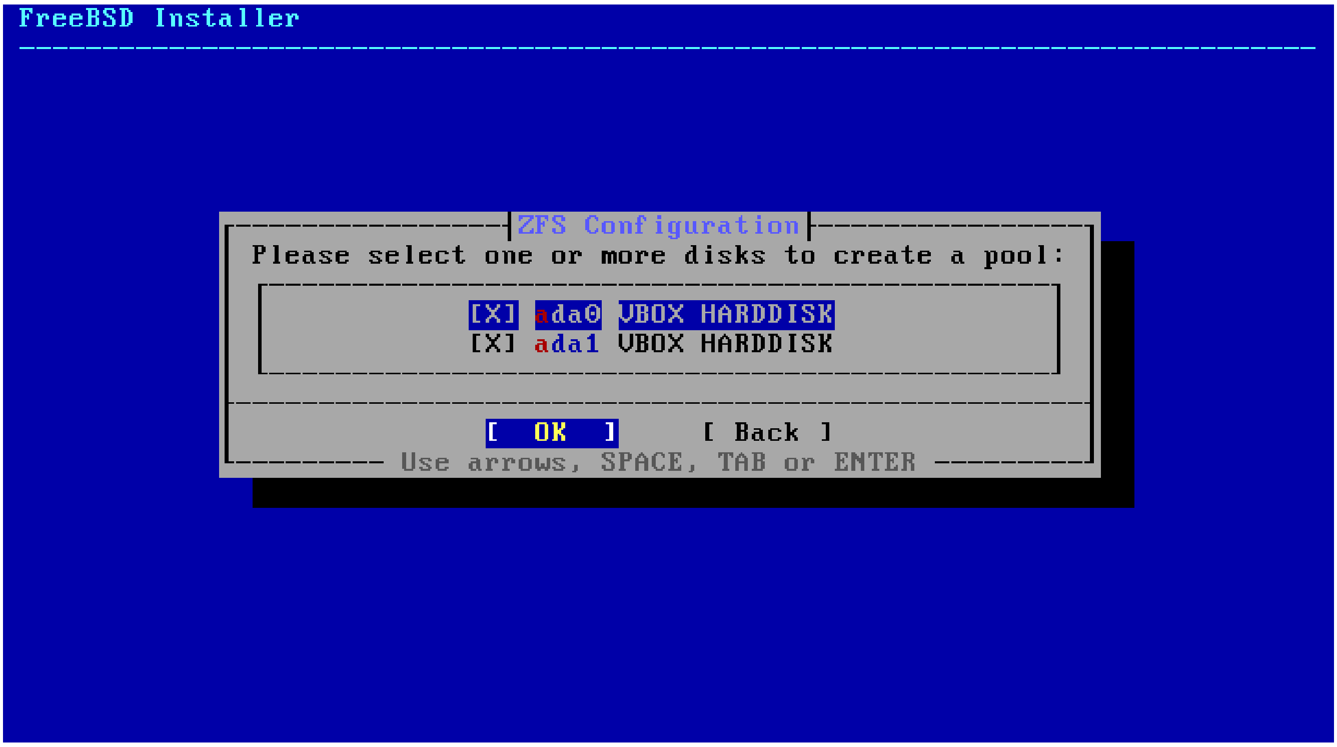
```
FreeBSD Installer
------------------------------------------------------------------------

                ┌──────────────┤ZFS Configuration├─────────────┐
                │ Select Virtual Device type:                   │
                │  ┌─────────────────────────────────────────┐  │
                │  │ stripe Stripe - No Redundancy            │  │
                │  │ mirror Mirror - n-Way Mirroring          │  │
                │  │ raid10 RAID 1+0 - n x 2-Way Mirrors      │  │
                │  │ raidz1 RAID-Z1 - Single Redundant RAID   │  │
                │  │ raidz2 RAID-Z2 - Double Redundant RAID   │  │
                │  │ raidz3 RAID-Z3 - Triple Redundant RAID   │  │
                │  └─────────────────────────────────────────┘  │
                ├───────────────────────────────────────────────┤
                │             [  OK  ]     [Cancel]             │
                └──────── Press arrows, TAB or ENTER ───────────┘

 [1+ Disks] Striping provides maximum storage but no redundancy
```

Figure 21. ZFS Pool Type

Here is a summary of the `Pool Type` that can be selected in this menu:

- `stripe` - Striping provides maximum storage of all connected devices, but no redundancy. If just one disk fails the data on the pool is lost irrevocably.
- `mirror` - Mirroring stores a complete copy of all data on every disk. Mirroring provides good read performance because data is read from all disks in parallel. Write performance is slower as the data must be written to all disks in the pool. Allows all but one disk to fail. This option requires at least two disks.
- `raid10` - Striped mirrors. Provides the best performance, but the least storage. This option needs at least an even number of disks and a minimum of four disks.
- `raidz1` - Single Redundant RAID. Allow one disk to fail concurrently. This option needs at least three disks.
- `raidz2` - Double Redundant RAID. Allows two disks to fail concurrently. This option needs at least four disks.
- `raidz3` - Triple Redundant RAID. Allows three disks to fail concurrently. This option needs at least five disks.

Once a `Pool Type` has been selected, a list of available disks is displayed, and the user is prompted to select one or more disks to make up the pool. The configuration is then validated to ensure that enough disks are selected. If validation fails, select **[<Change Selection>]** to return to the list of disks or **[<Back>]** to change the `Pool Type`.



FreeBSD Installer

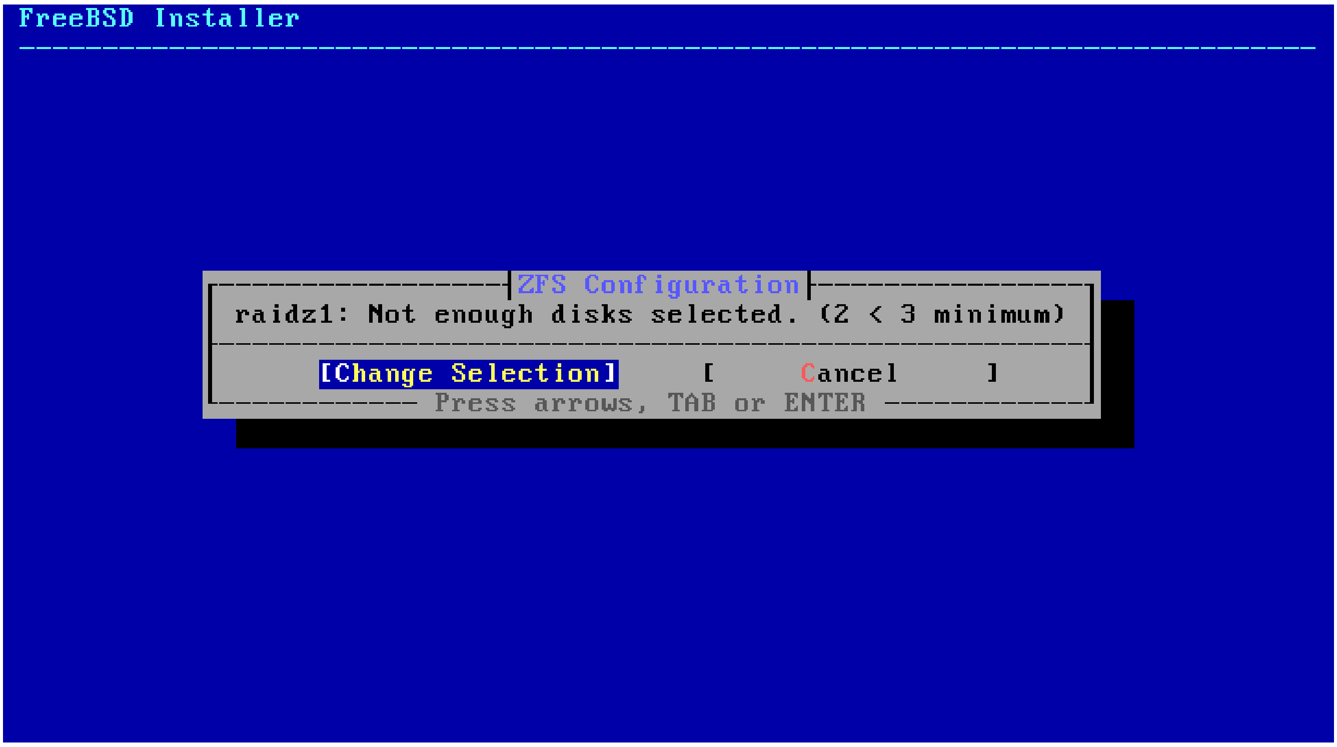
ZFS Configuration

Please select one or more disks to create a pool:

```
[X] ada0 VBOX HARDDISK
[X] ada1 VBOX HARDDISK
```

[OK]
[Back]
Use arrows, SPACE, TAB or ENTER

Figure 22. Disk Selection



FreeBSD Installer

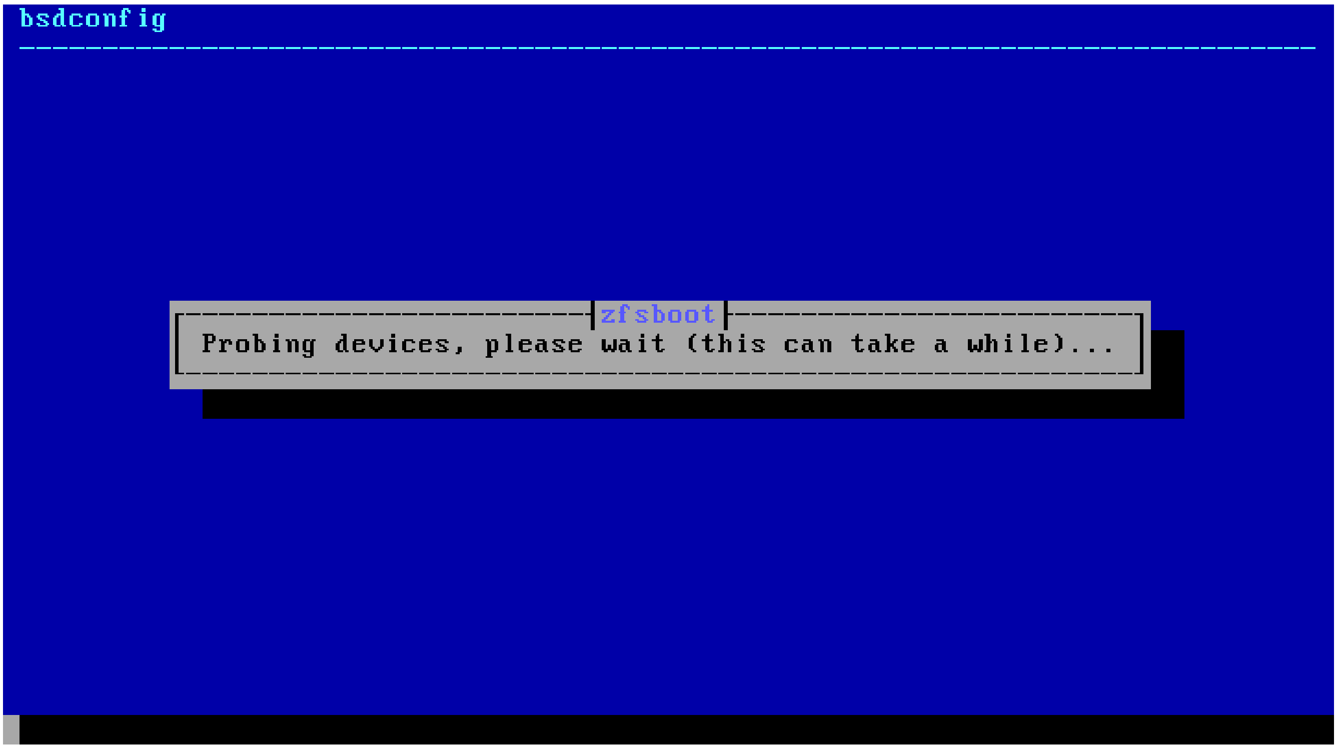
ZFS Configuration

```
raidz1: Not enough disks selected. (2 < 3 minimum)
```

[Change Selection]
[Cancel]
Press arrows, TAB or ENTER

Figure 23. Invalid Selection

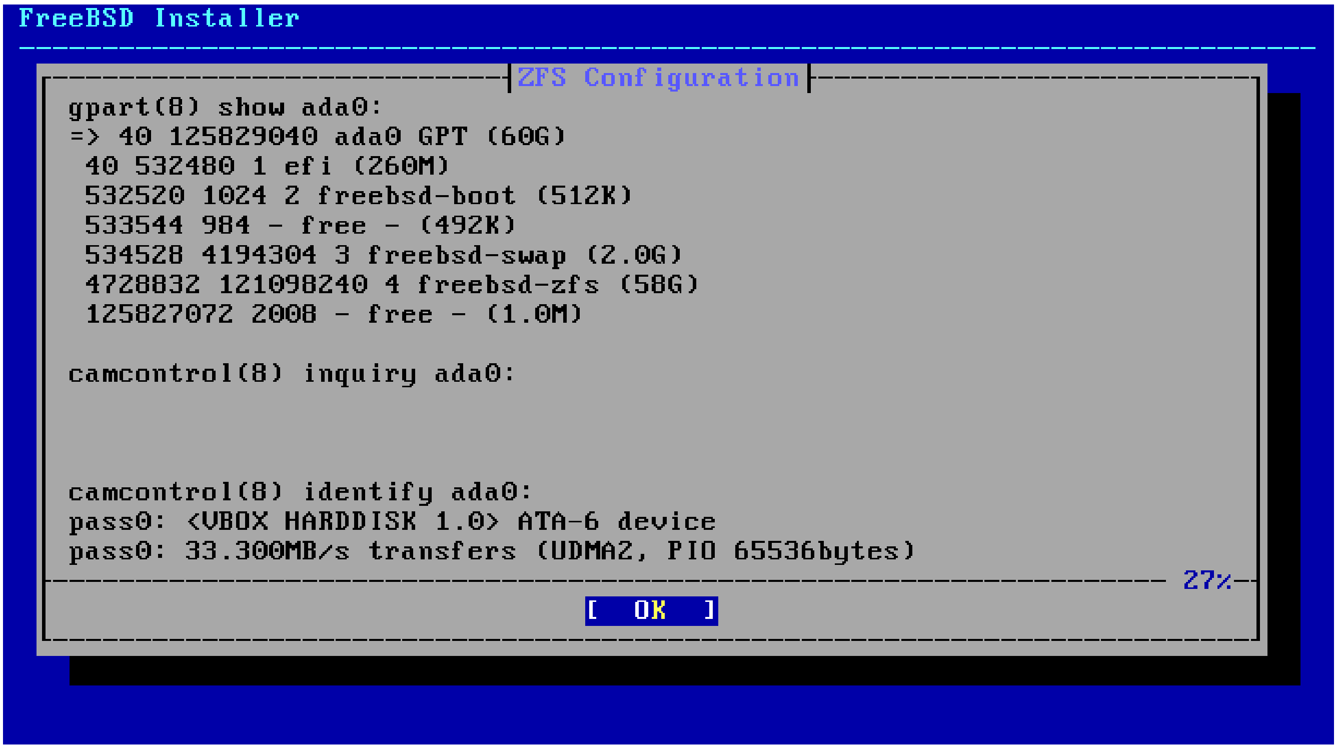
If one or more disks are missing from the list, or if disks were attached after the installer was started, select **[- Rescan Devices]** to repopulate the list of available disks.



```
Probing devices, please wait (this can take a while)...
```

Figure 24. Rescan Devices

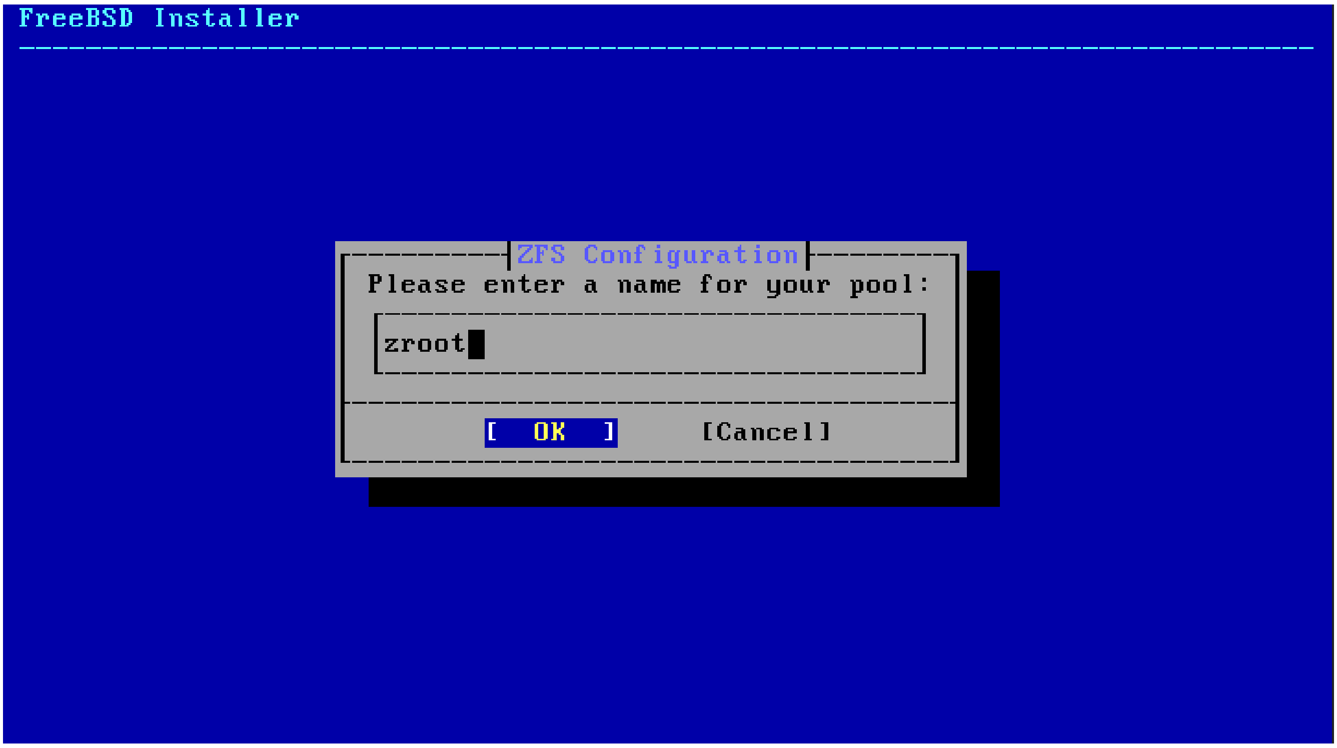
To avoid accidentally erasing the wrong disk, the **[- Disk Info]** menu can be used to inspect each disk, including its partition table and various other information such as the device model number and serial number, if available.



```
gpart(8) show ada0:
=> 40 125829040 ada0 GPT (60G)
 40 532480 1 efi (260M)
 532520 1024 2 freebsd-boot (512K)
 533544 984 - free - (492K)
 534528 4194304 3 freebsd-swap (2.0G)
 4728832 121098240 4 freebsd-zfs (58G)
 125827072 2008 - free - (1.0M)

camcontrol(8) inquiry ada0:



camcontrol(8) identify ada0:
pass0: <VBOX HARDDISK 1.0> ATA-6 device
pass0: 33.300MB/s transfers (UDMA2, PIO 65536bytes)
```

Figure 25. Analyzing a Disk

Select `N` to configure the `Pool Name`. Enter the desired name, then select **[<OK>]** to establish it or **[<Cancel>]** to return to the main menu and leave the default name.


FreeBSD Installer

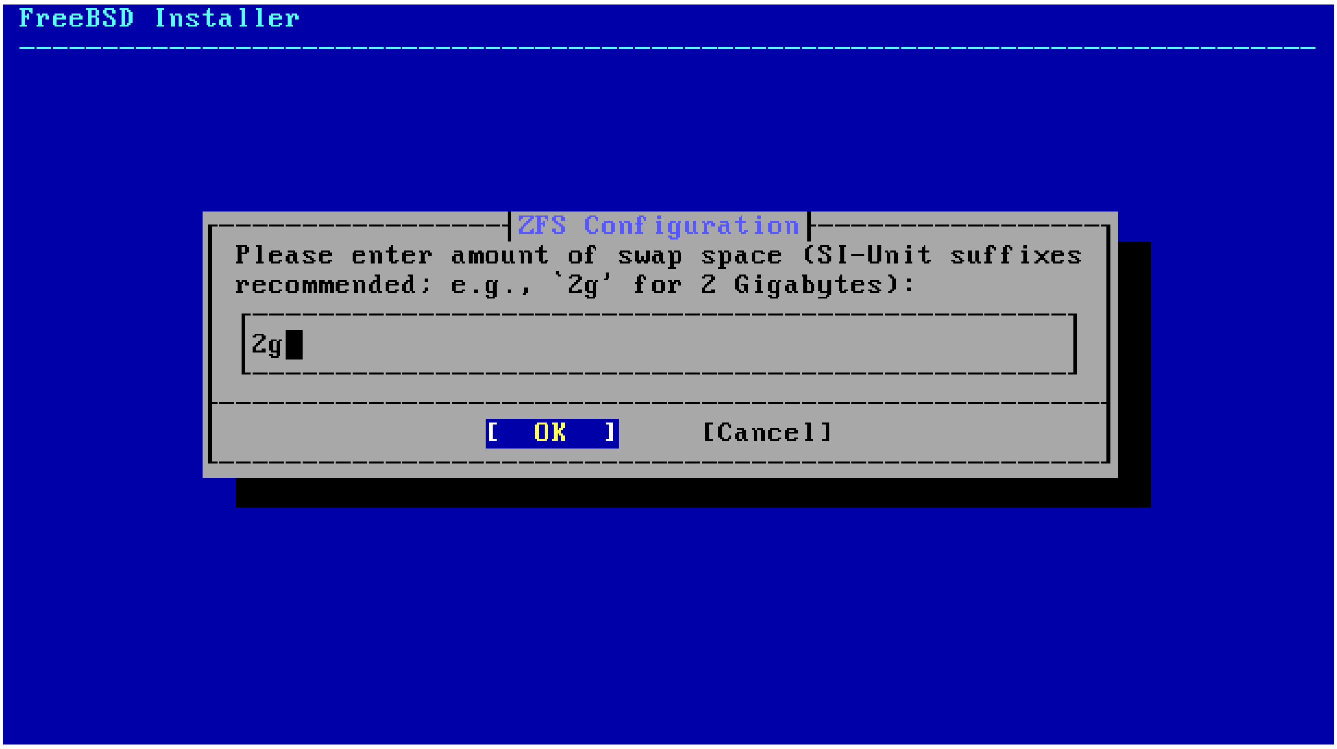
ZFS Configuration

Please enter a name for your pool:

zroot

Figure 26. Pool Name

Select S to set the amount of swap. Enter the desired amount of swap, then select **[<OK>]** to establish it or **[<Cancel>]** to return to the main menu and let the default amount.


FreeBSD Installer

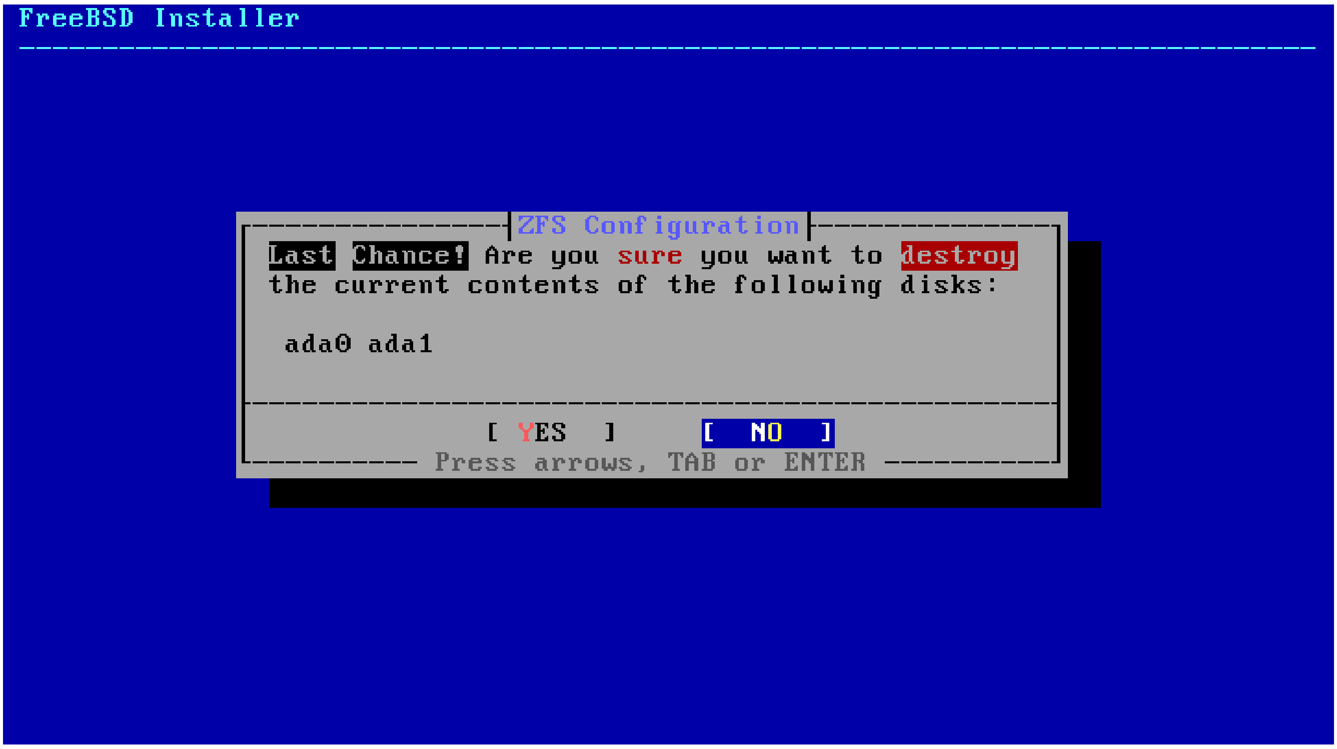
ZFS Configuration

Please enter amount of swap space (SI-Unit suffixes recommended; e.g., `2g' for 2 Gigabytes):

2g

Figure 27. Swap Amount

Once all options have been set to the desired values, select the **[>>> Install]** option at the top of the menu. The installer then offers a last chance to cancel before the contents of the selected drives are destroyed to create the ZFS pool.


FreeBSD Installer

ZFS Configuration

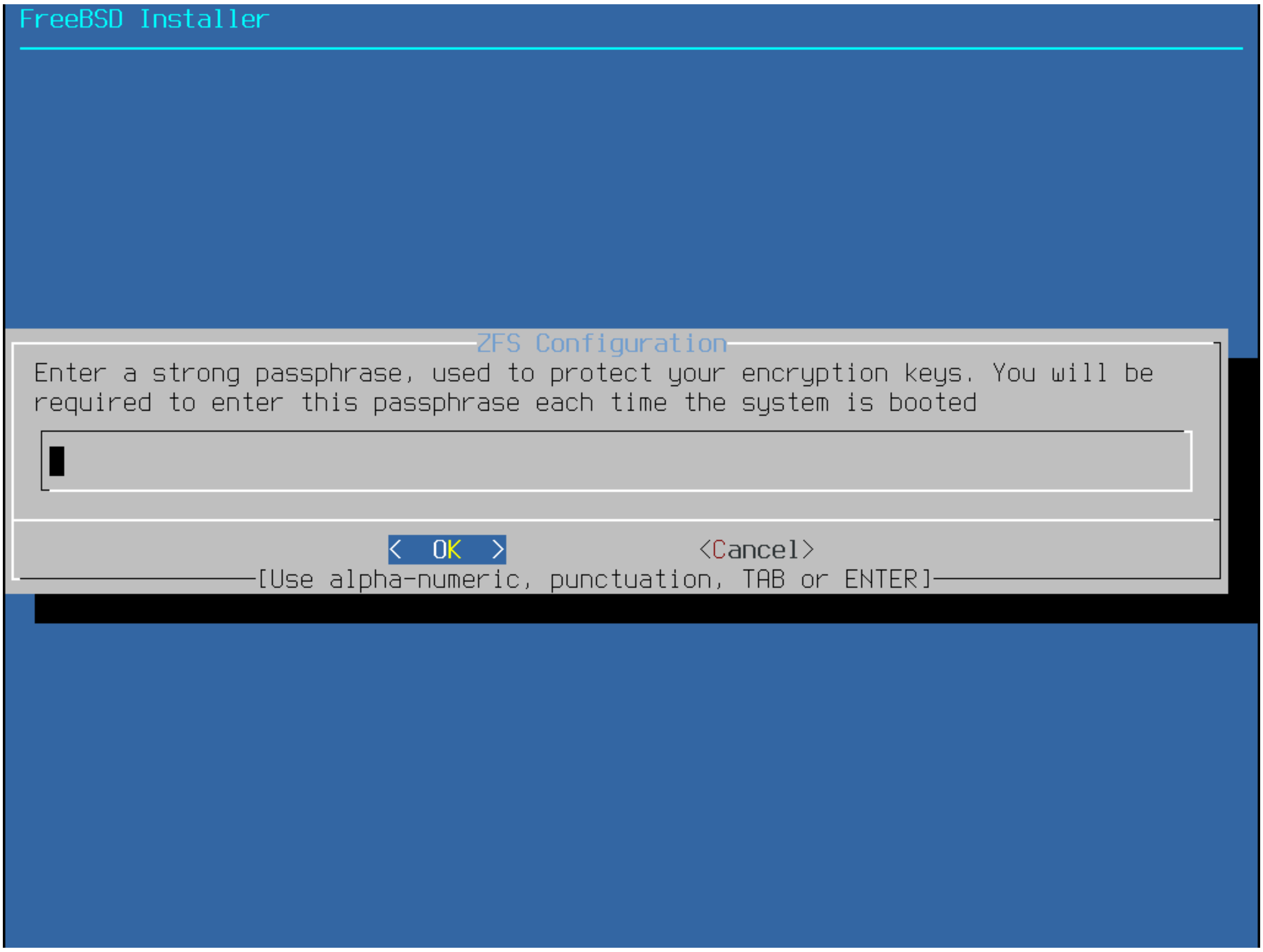
Last Chance! Are you sure you want to destroy the current contents of the following disks:

ada0 ada1

[YES]
[NO]
Press arrows, TAB or ENTER

Figure 28. Last Chance

If GELI disk encryption was enabled, the installer will prompt twice for the passphrase to be used to encrypt the disks. Initialization of the encryption then begins.

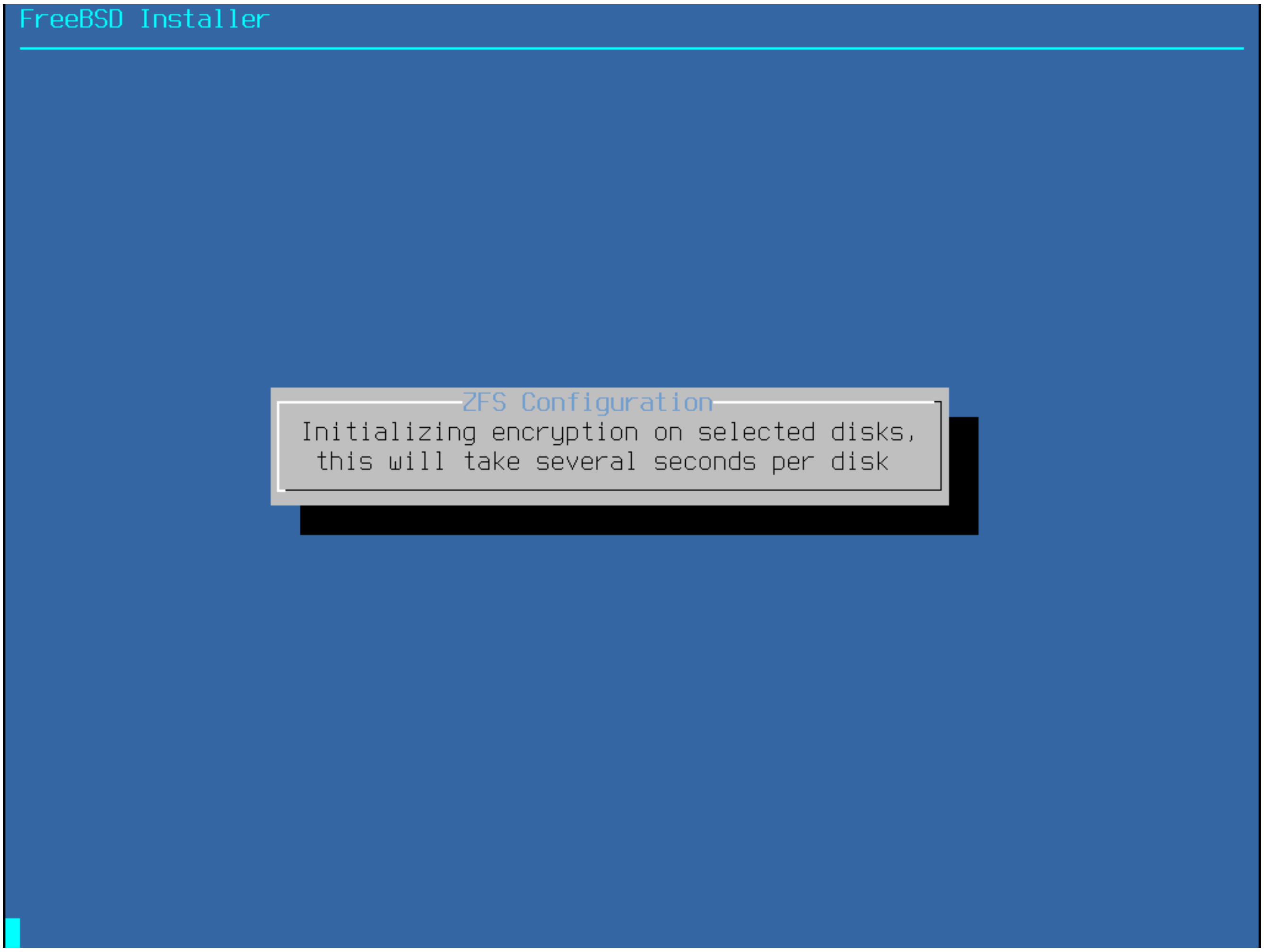

FreeBSD Installer

ZFS Configuration

Enter a strong passphrase, used to protect your encryption keys. You will be required to enter this passphrase each time the system is booted

< OK >
<Cancel>
[Use alpha-numeric, punctuation, TAB or ENTER]

Figure 29. Disk Encryption Password



Initializing encryption on selected disks, this will take several seconds per disk

Figure 30. Initializing Encryption

The installation then proceeds normally. To continue with the installation, go to Fetching Distribution Files.

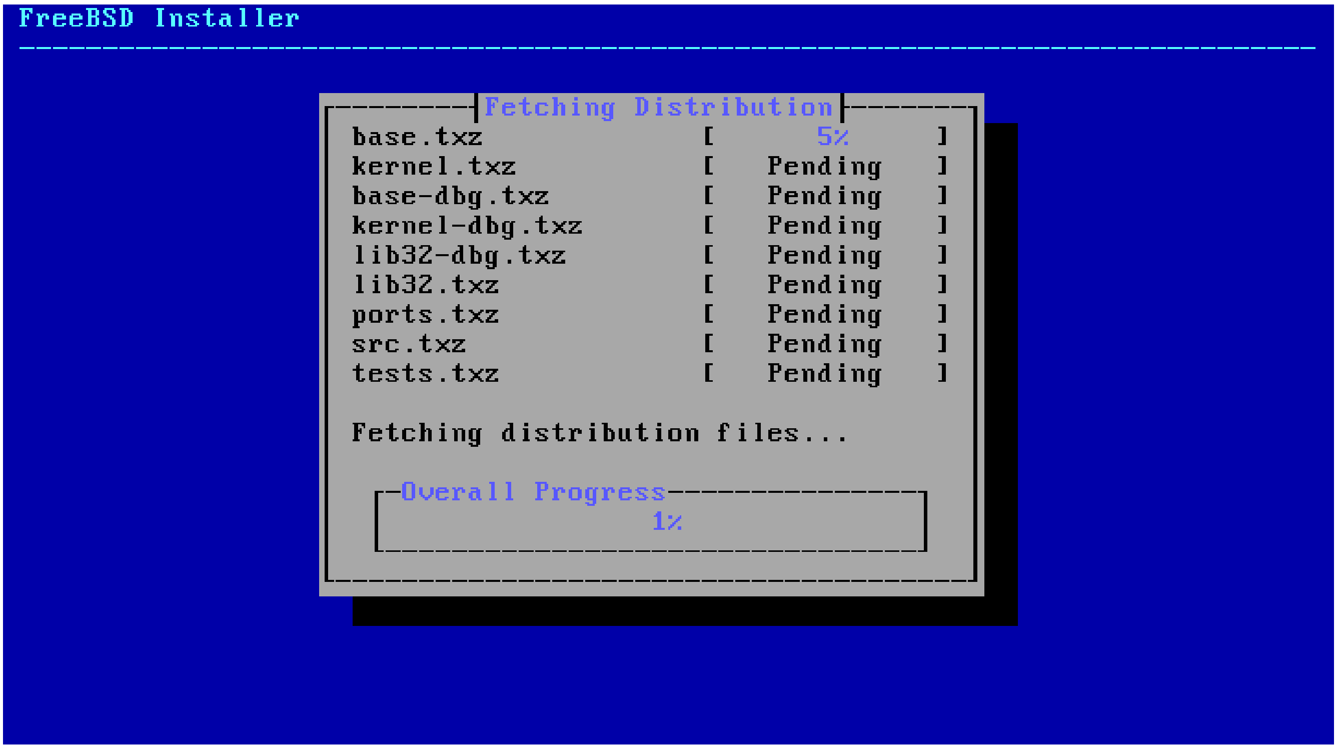
2.6.5. Shell Mode Partitioning

When creating advanced installations, the bsdinstall partitioning menus may not provide the level of flexibility required. Advanced users can select the **[Shell]** option from the partitioning menu in order to manually partition the drives, create the file system(s), populate `/tmp/bsdinstall_etc/fstab`, and mount the file systems under `/mnt`. Once this is done, type `exit` to return to bsdinstall and continue the installation.

2.7. Fetching Distribution Files

Installation time will vary depending on the distributions chosen, installation media, and speed of the computer. A series of messages will indicate the progress.

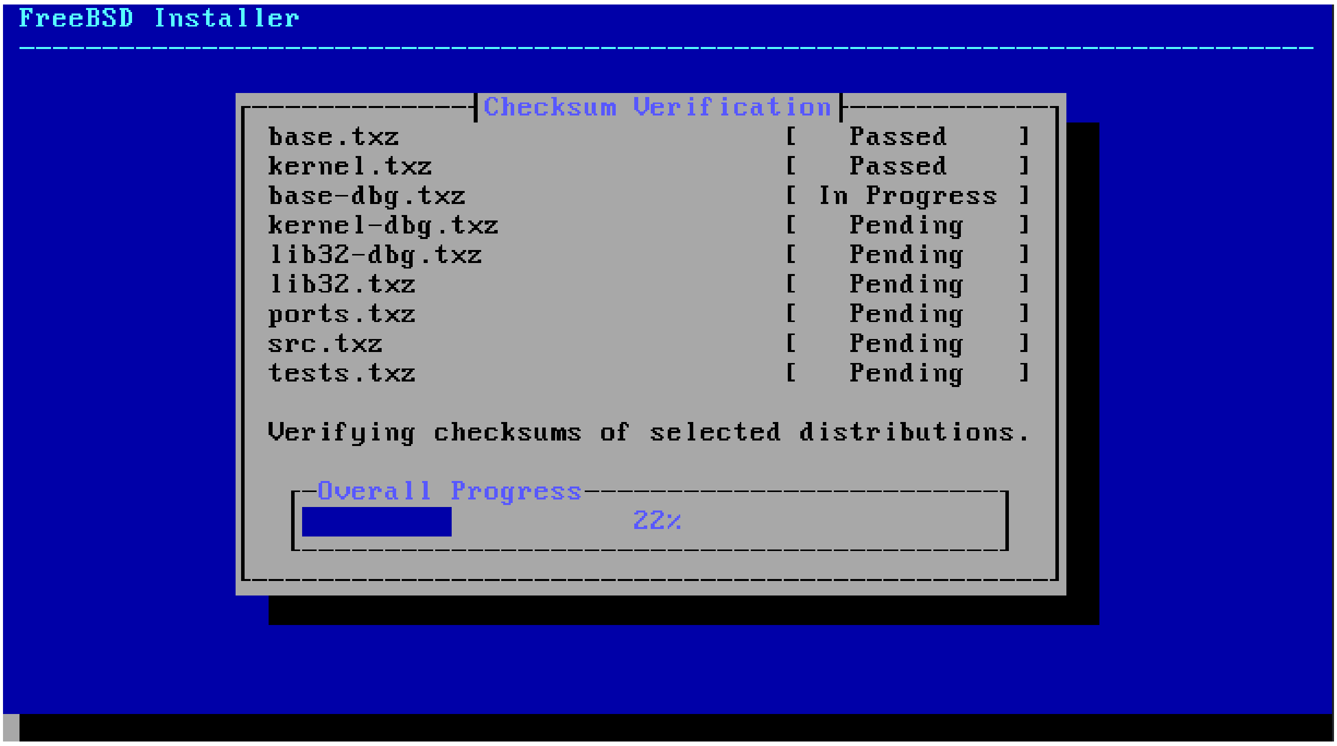
First, the installer formats the selected disk(s) and initializes the partitions. Next, in the case of a `bootonly media` or `mini memstick`, it downloads the selected components:



```
FreeBSD Installer
--------------------------------------------------------------------------------

            ┌---------┤Fetching Distribution├--------┐
            │ base.txz                [      5%     ] │
            │ kernel.txz              [   Pending   ] │
            │ base-dbg.txz            [   Pending   ] │
            │ kernel-dbg.txz          [   Pending   ] │
            │ lib32-dbg.txz           [   Pending   ] │
            │ lib32.txz               [   Pending   ] │
            │ ports.txz               [   Pending   ] │
            │ src.txz                 [   Pending   ] │
            │ tests.txz               [   Pending   ] │
            │                                         │
            │ Fetching distribution files...          │
            │                                         │
            │   ┌Overall Progress-----------------┐   │
            │   │                1%               │   │
            │   └---------------------------------┘   │
            └-----------------------------------------┘
```

Figure 31. Fetching Distribution Files

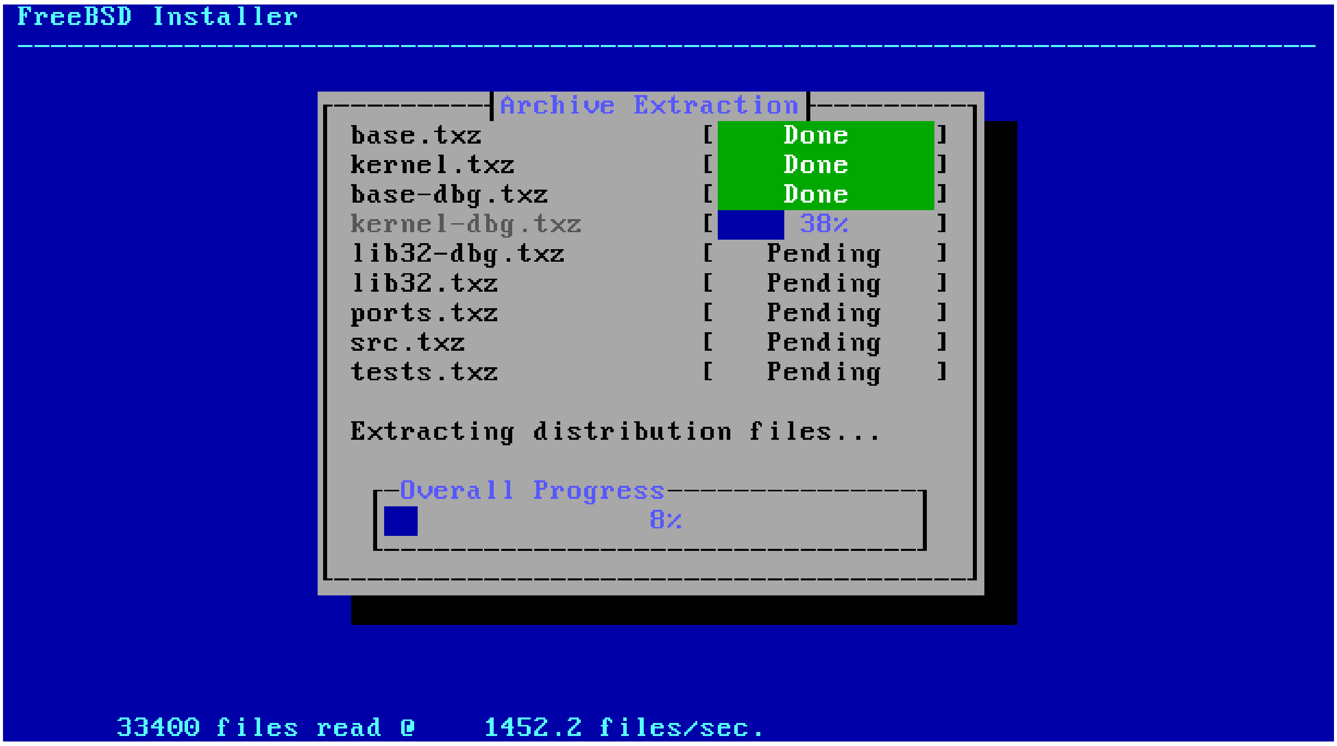
Next, the integrity of the distribution files is verified to ensure they have not been corrupted during download or misread from the installation media:



```
FreeBSD Installer
--------------------------------------------------------------------------------

        ┌-------------┤Checksum Verification├------------┐
        │ base.txz                         [    Passed     ] │
        │ kernel.txz                       [    Passed     ] │
        │ base-dbg.txz                     [  In Progress  ] │
        │ kernel-dbg.txz                   [    Pending    ] │
        │ lib32-dbg.txz                    [    Pending    ] │
        │ lib32.txz                        [    Pending    ] │
        │ ports.txz                        [    Pending    ] │
        │ src.txz                          [    Pending    ] │
        │ tests.txz                        [    Pending    ] │
        │                                                  │
        │ Verifying checksums of selected distributions.   │
        │                                                  │
        │   ┌Overall Progress--------------------------┐   │
        │   │                  22%                     │   │
        │   └------------------------------------------┘   │
        └--------------------------------------------------┘
```

Figure 32. Verifying Distribution Files

Finally, the verified distribution files are extracted to the disk:



```
FreeBSD Installer
--------------------------------------------------------------------------------

                   ┌──────────┤Archive Extraction├─────────┐
                   │ base.txz                [    Done    ]│
                   │ kernel.txz              [    Done    ]│
                   │ base-dbg.txz            [    Done    ]│
                   │ kernel-dbg.txz          [     38%    ]│
                   │ lib32-dbg.txz           [  Pending   ]│
                   │ lib32.txz               [  Pending   ]│
                   │ ports.txz               [  Pending   ]│
                   │ src.txz                 [  Pending   ]│
                   │ tests.txz               [  Pending   ]│
                   │                                       │
                   │ Extracting distribution files...      │
                   │                                       │
                   │   ┌Overall Progress───────────────┐   │
                   │   │            8%                 │   │
                   │   └───────────────────────────────┘   │
                   └───────────────────────────────────────┘

      33400 files read @     1452.2 files/sec.
```

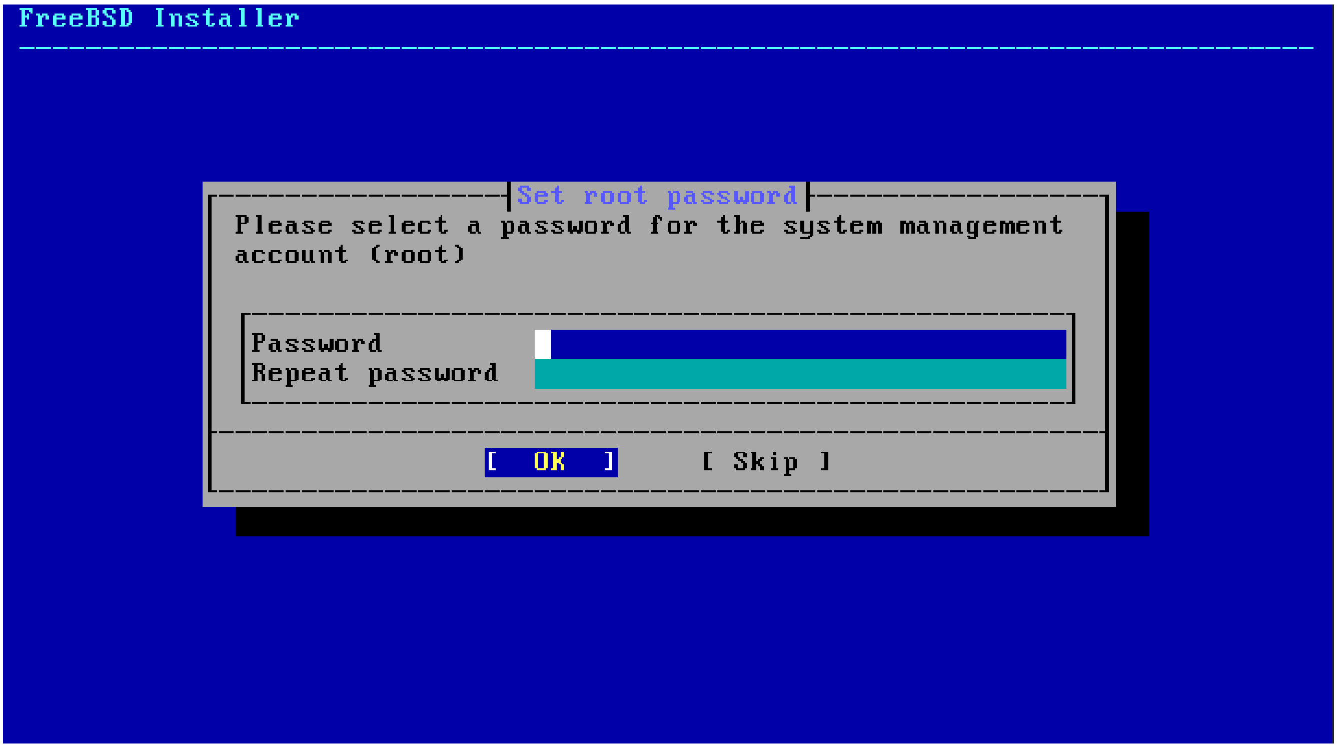
Figure 33. Extracting Distribution Files

Once all requested distribution files have been extracted, bsdinstall displays the first post-installation configuration screen. The available post-configuration options are described in the next section.

2.8. Network Interfaces, Accounts, Time Zone, Services and Hardening

2.8.1. Setting the `root` Password

First, the `root` password must be set. While entering the password, the characters being typed are not displayed on the screen. The password must be entered twice to prevent typing errors.

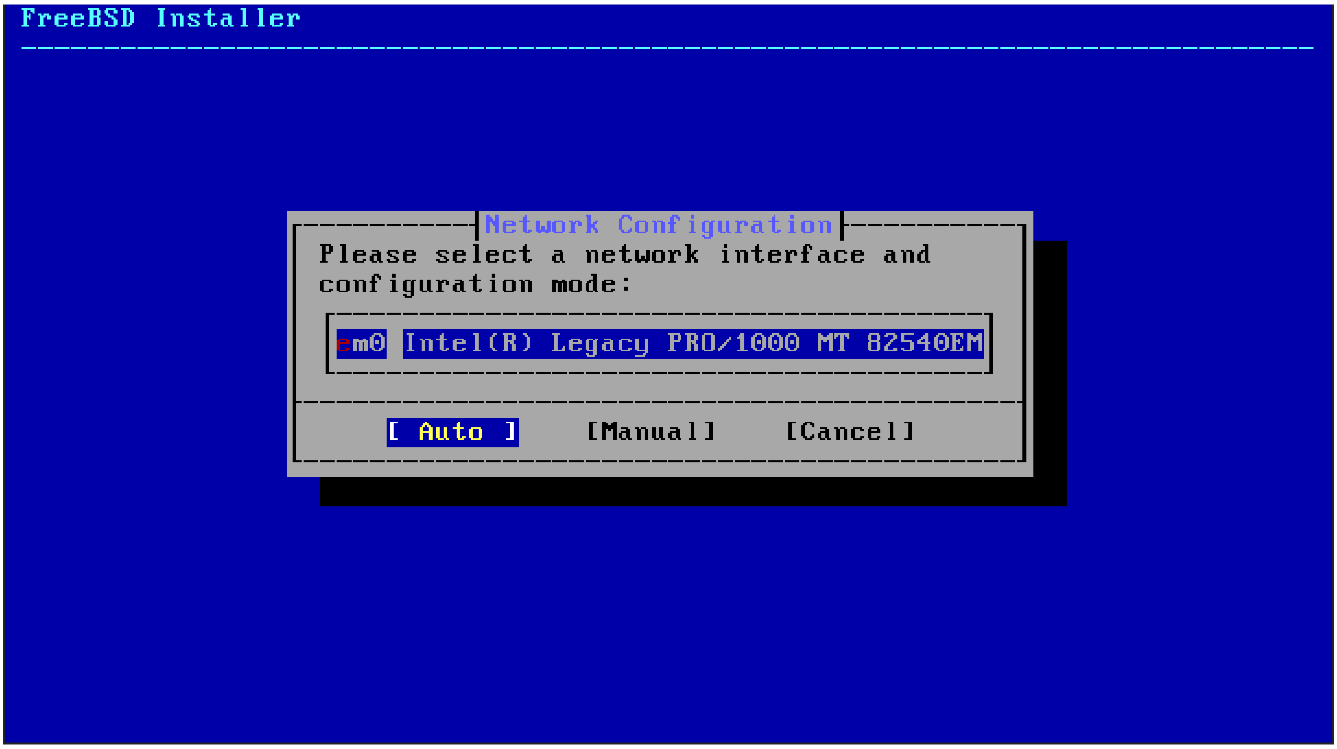

FreeBSD Installer
Set root password
Password
Repeat password
[OK]
[Skip]

Please select a password for the system management account (root)

Figure 34. Setting the `root` Password

2.8.2. Configuring Network Interfaces

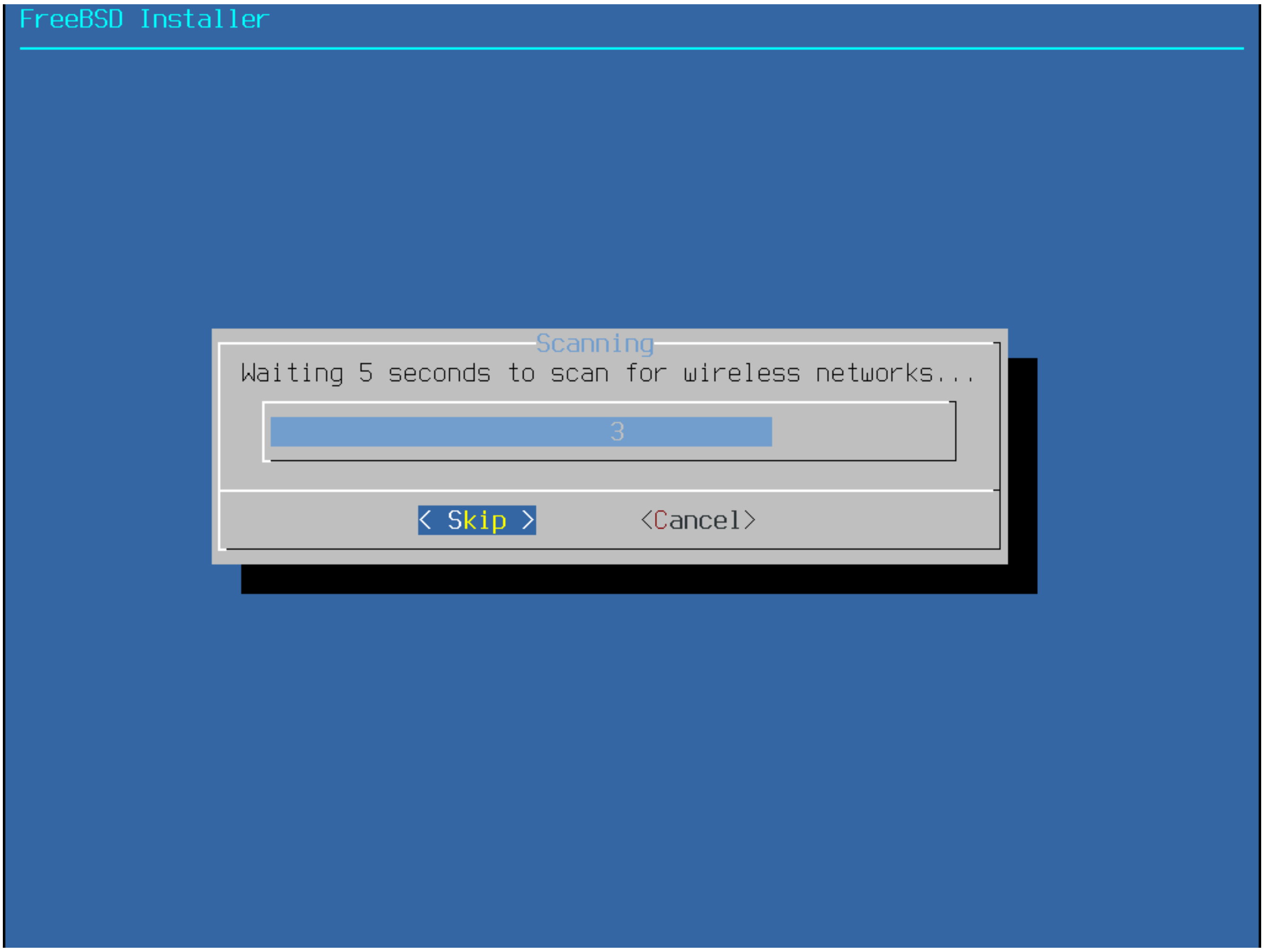
Next, a list of the network interfaces found on the computer is shown. Select the interface to configure.


FreeBSD Installer
Network Configuration
em0 Intel(R) Legacy PRO/1000 MT 82540EM
[Auto]
[Manual]
[Cancel]

Please select a network interface and configuration mode:

Figure 35. Choose a Network Interface

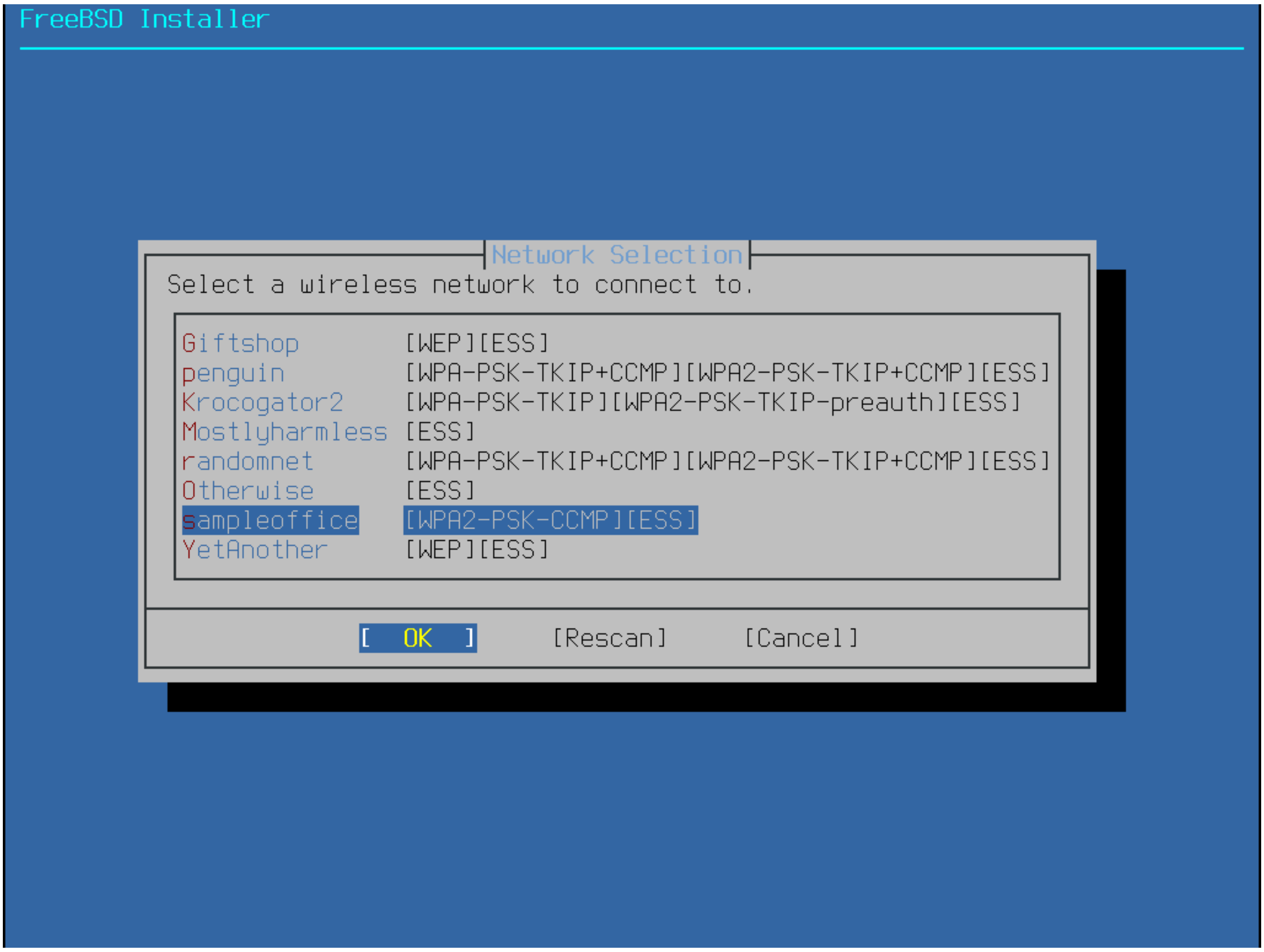
If an Ethernet interface is selected, the installer will skip ahead to the menu shown in Choose IPv4 Networking. If a wireless network interface is chosen, the system will instead scan for wireless access points:



Waiting 5 seconds to scan for wireless networks...

Figure 36. Scanning for Wireless Access Points

Wireless networks are identified by a Service Set Identifier (SSID); a short, unique name given to each network. SSIDs found during the scan are listed, followed by a description of the encryption types available for that network. If the desired SSID does not appear in the list, select **[Rescan]** to scan again. If the desired network still does not appear, check for problems with antenna connections or try moving the computer closer to the access point. Rescan after each change is made.



```
Select a wireless network to connect to.

Giftshop       [WEP][ESS]
penguin        [WPA-PSK-TKIP+CCMP][WPA2-PSK-TKIP+CCMP][ESS]
Krocogator2    [WPA-PSK-TKIP][WPA2-PSK-TKIP-preauth][ESS]
Mostlyharmless [ESS]
randomnet      [WPA-PSK-TKIP+CCMP][WPA2-PSK-TKIP+CCMP][ESS]
Otherwise      [ESS]
sampleoffice   [WPA2-PSK-CCMP][ESS]
YetAnother     [WEP][ESS]
```

Figure 37. Choosing a Wireless Network

Next, enter the encryption information for connecting to the selected wireless network. WPA2 encryption is strongly recommended over older encryption types such as WEP, which offer little security. If the network uses WPA2, input the password, also known as the Pre-Shared Key (PSK). For security reasons, the characters typed into the input box are displayed as asterisks.

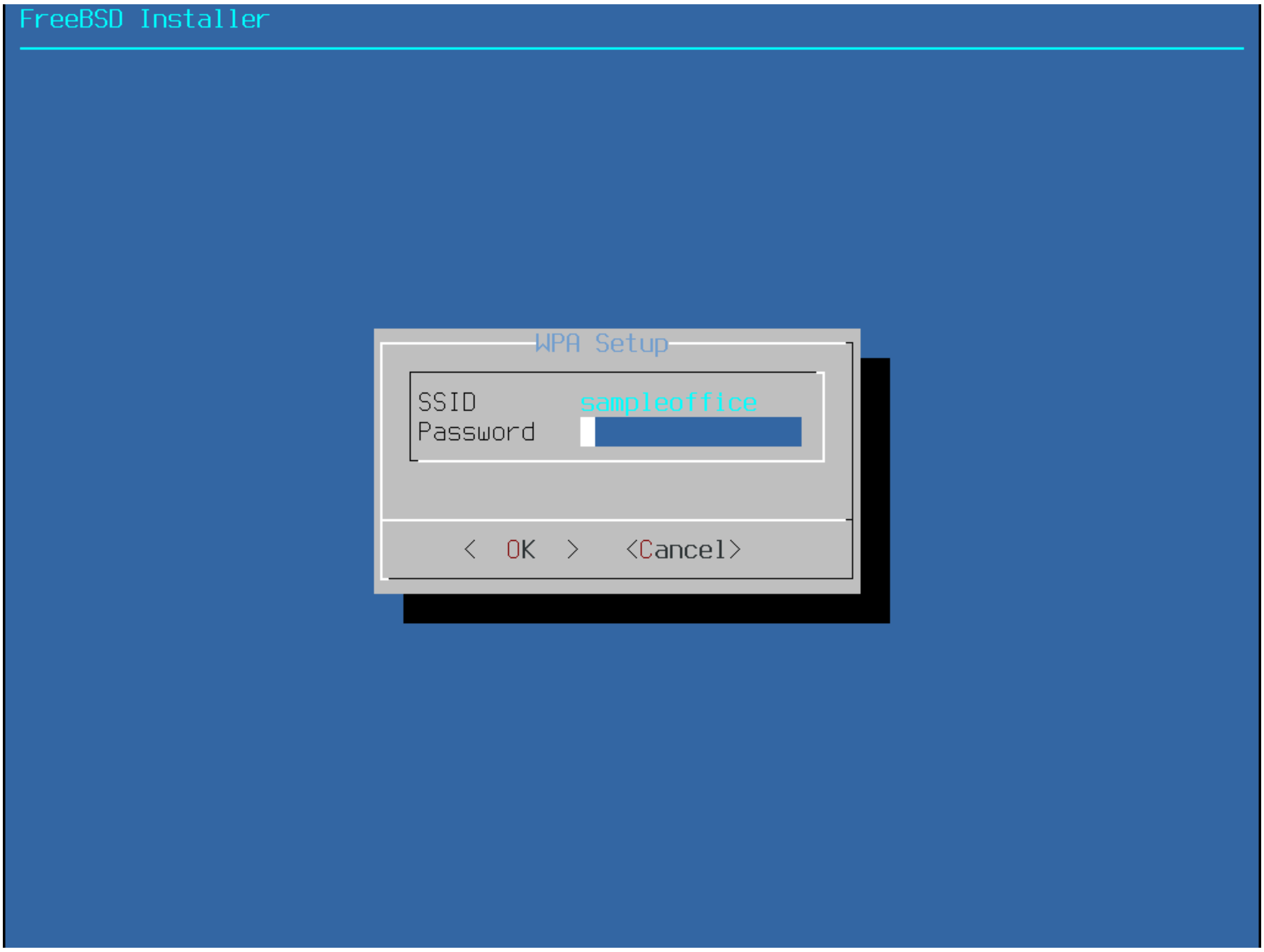
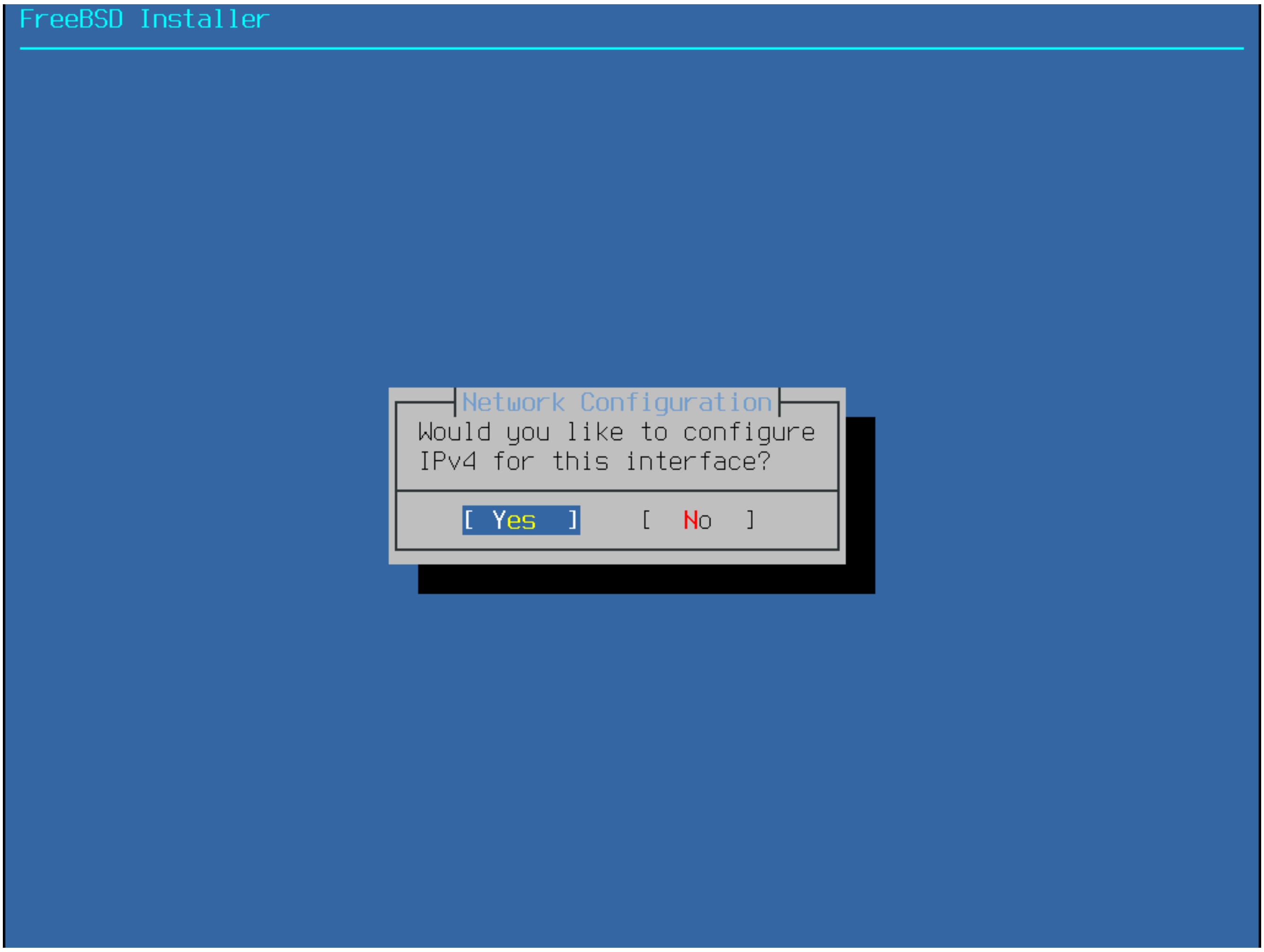

FreeBSD Installer
WPA Setup
SSID
sampleoffice
Password
< OK >
<Cancel>

Figure 38. WPA2 Setup

Next, choose whether or not an IPv4 address should be configured on the Ethernet or wireless interface:



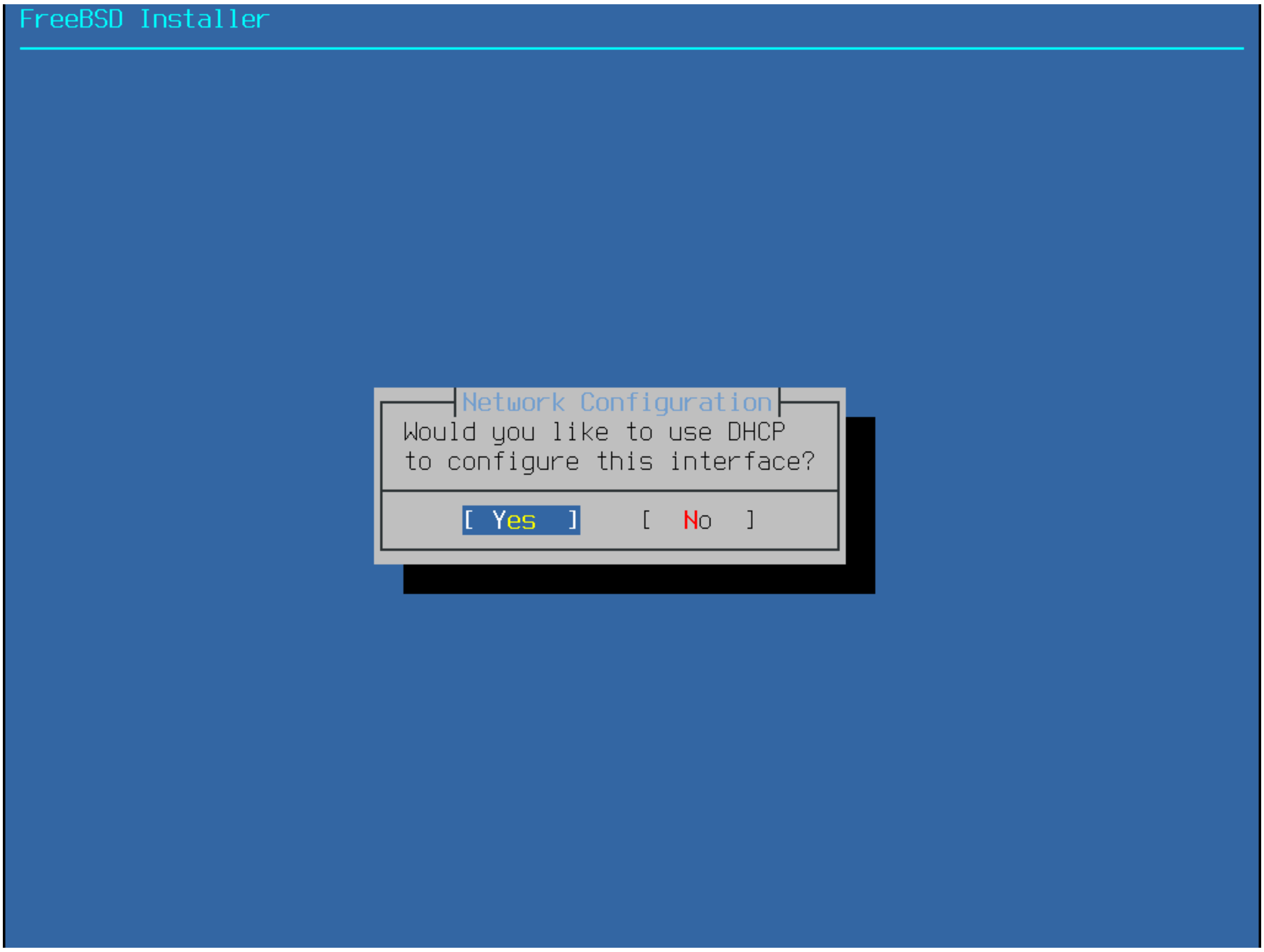
Would you like to configure IPv4 for this interface?

Figure 39. Choose IPv4 Networking

There are two methods of IPv4 configuration. DHCP will automatically configure the network interface correctly and should be used if the network provides a DHCP server. Otherwise, the addressing information needs to be input manually as a static configuration.

> Do not enter random network information as it will not work. If a DHCP server is not available, obtain the information listed in Required Network Information from the network administrator or Internet service provider.

If a DHCP server is available, select **[Yes]** in the next menu to automatically configure the network interface. The installer will appear to pause for a minute or so as it finds the DHCP server and obtains the addressing information for the system.

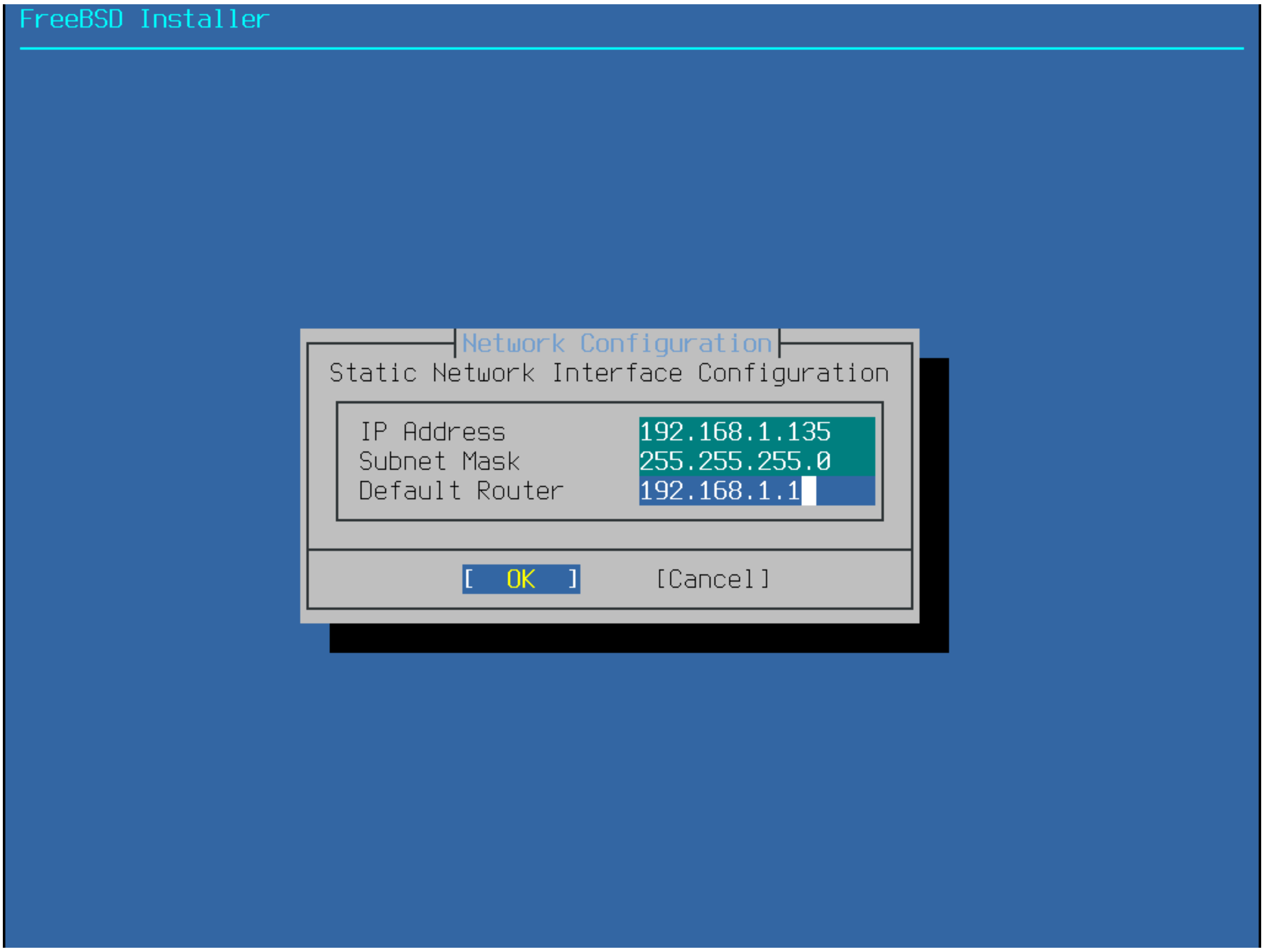

FreeBSD Installer

Network Configuration

Would you like to use DHCP to configure this interface?

Figure 40. Choose IPv4 DHCP Configuration

If a DHCP server is not available, select **[No]** and input the following addressing information in this menu:



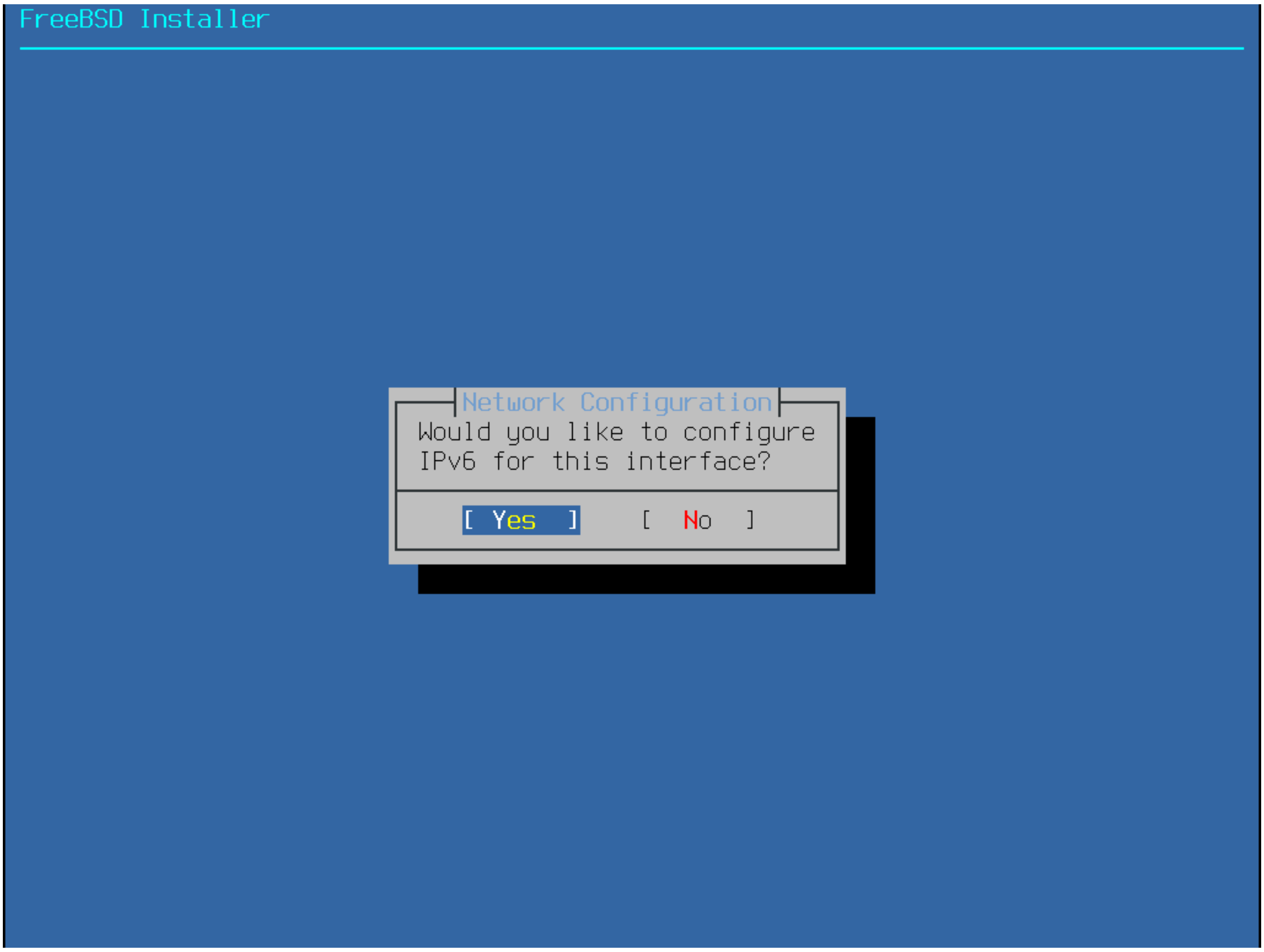
Static Network Interface Configuration

```
IP Address        192.168.1.135
Subnet Mask       255.255.255.0
Default Router    192.168.1.1
```

Figure 41. IPv4 Static Configuration

- `IP Address` - The IPv4 address assigned to this computer. The address must be unique and not already in use by another device on the local network.
- `Subnet Mask` - The subnet mask for the network.
- `Default Router` - The IP address of the network's default gateway.

The next screen will ask if the interface should be configured for IPv6. If IPv6 is available and desired, choose **[Yes]** to select it.

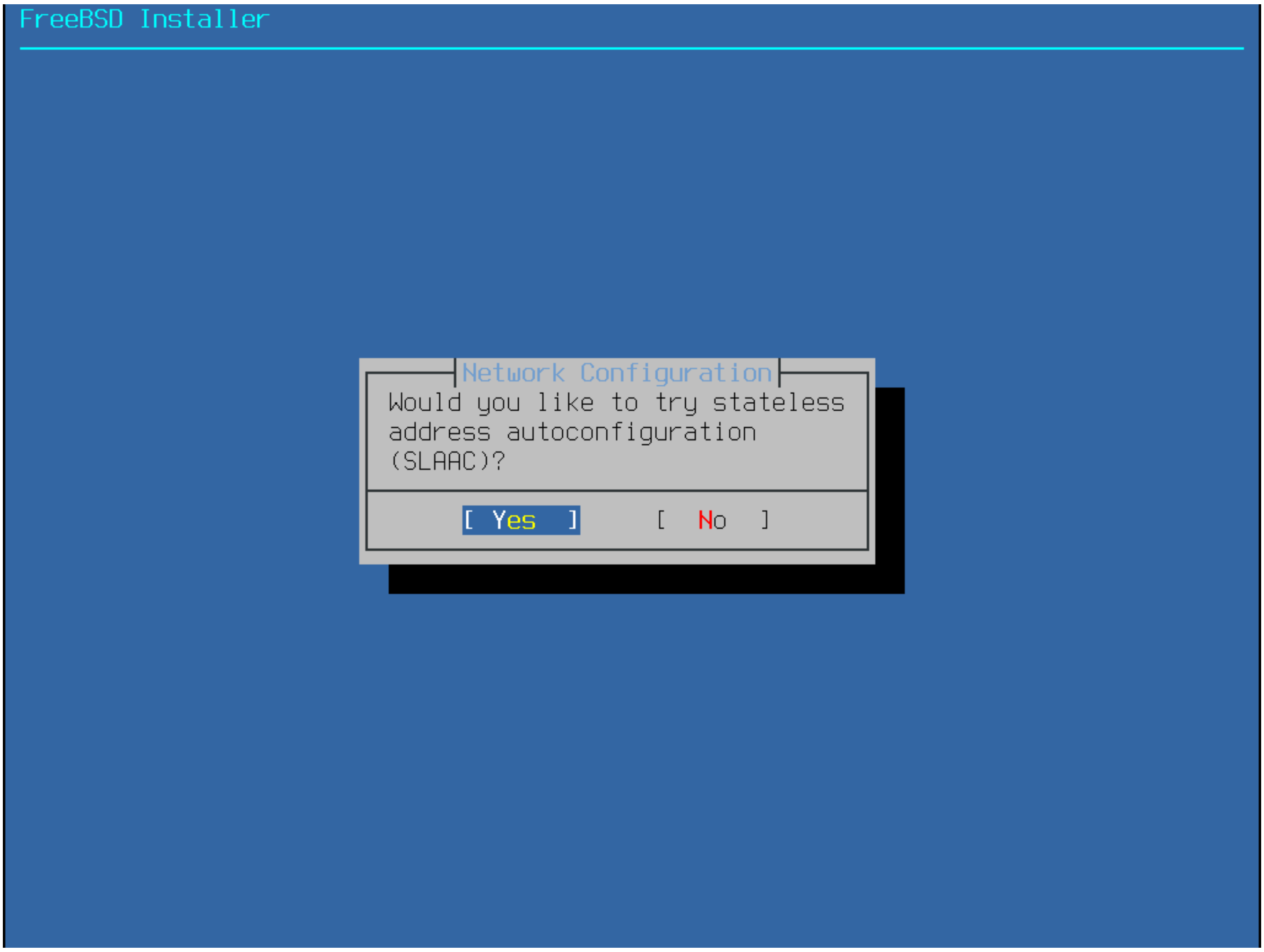


Would you like to configure IPv6 for this interface?

Figure 42. Choose IPv6 Networking

IPv6 also has two methods of configuration. StateLess Address AutoConfiguration (SLAAC) will automatically request the correct configuration information from a local router. Refer to rfc4862 for more information. Static configuration requires manual entry of network information.

If an IPv6 router is available, select **[Yes]** in the next menu to automatically configure the network interface. The installer will appear to pause for a minute or so as it finds the router and obtains the addressing information for the system.



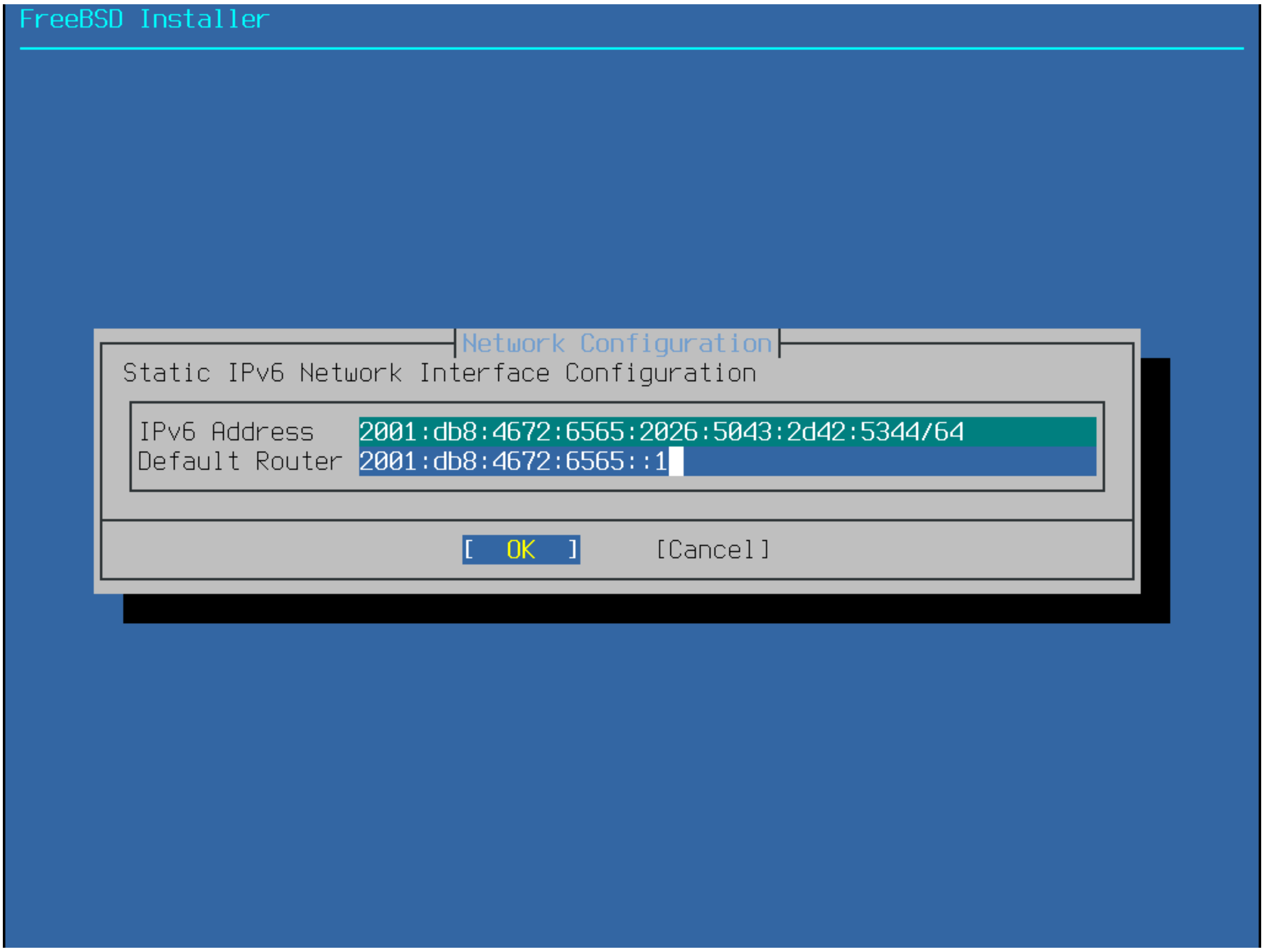
FreeBSD Installer

Network Configuration

Would you like to try stateless address autoconfiguration (SLAAC)?

Figure 43. Choose IPv6 SLAAC Configuration

If an IPv6 router is not available, select **[No]** and input the following addressing information in this menu:



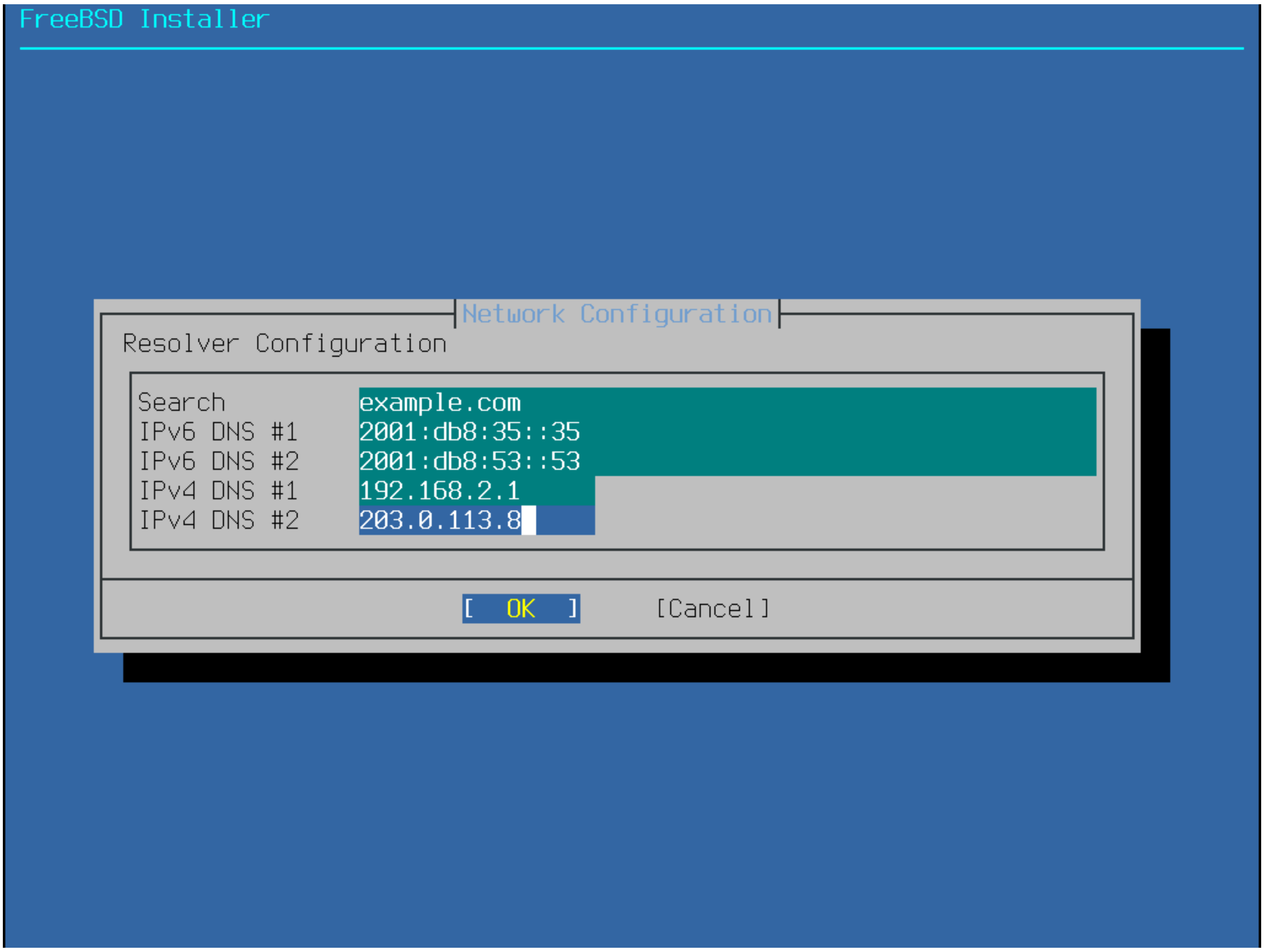
Static IPv6 Network Interface Configuration

IPv6 Address 2001:db8:4672:6565:2026:5043:2d42:5344/64
Default Router 2001:db8:4672:6565::1

Figure 44. IPv6 Static Configuration

- `IPv6 Address` - The IPv6 address assigned to this computer. The address must be unique and not already in use by another device on the local network.
- `Default Router` - The IPv6 address of the network's default gateway.

The last network configuration menu is used to configure the Domain Name System (DNS) resolver, which converts hostnames to and from network addresses. If DHCP or SLAAC was used to autoconfigure the network interface, the `Resolver Configuration` values may already be filled in. Otherwise, enter the local network's domain name in the `Search` field. `DNS #1` and `DNS #2` are the IPv4 and/or IPv6 addresses of the DNS servers. At least one DNS server is required.

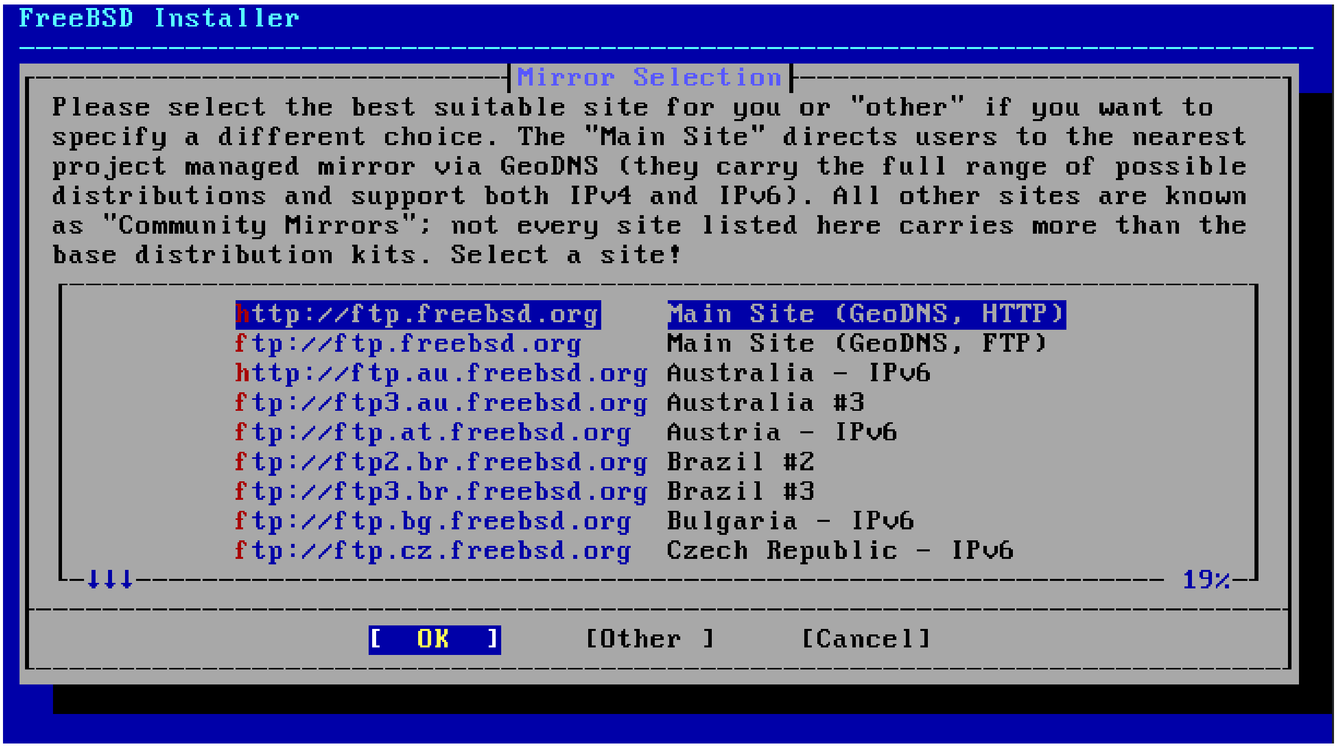


```
Resolver Configuration

Search          example.com
IPv6 DNS #1     2001:db8:35::35
IPv6 DNS #2     2001:db8:53::53
IPv4 DNS #1     192.168.2.1
IPv4 DNS #2     203.0.113.8
```

Figure 45. DNS Configuration

Once the interface is configured, select a mirror site that is located in the same region of the world as the computer on which FreeBSD is being installed. Files can be retrieved more quickly when the mirror is close to the target computer, reducing installation time.

Selecting `http://download.freebsd.org` `Main Site (GeoDNS, HTTP)` or `ftp://download.freebsd.org` `Main Site (GeoDNS, FTP)` will automatically route to the nearest mirror.



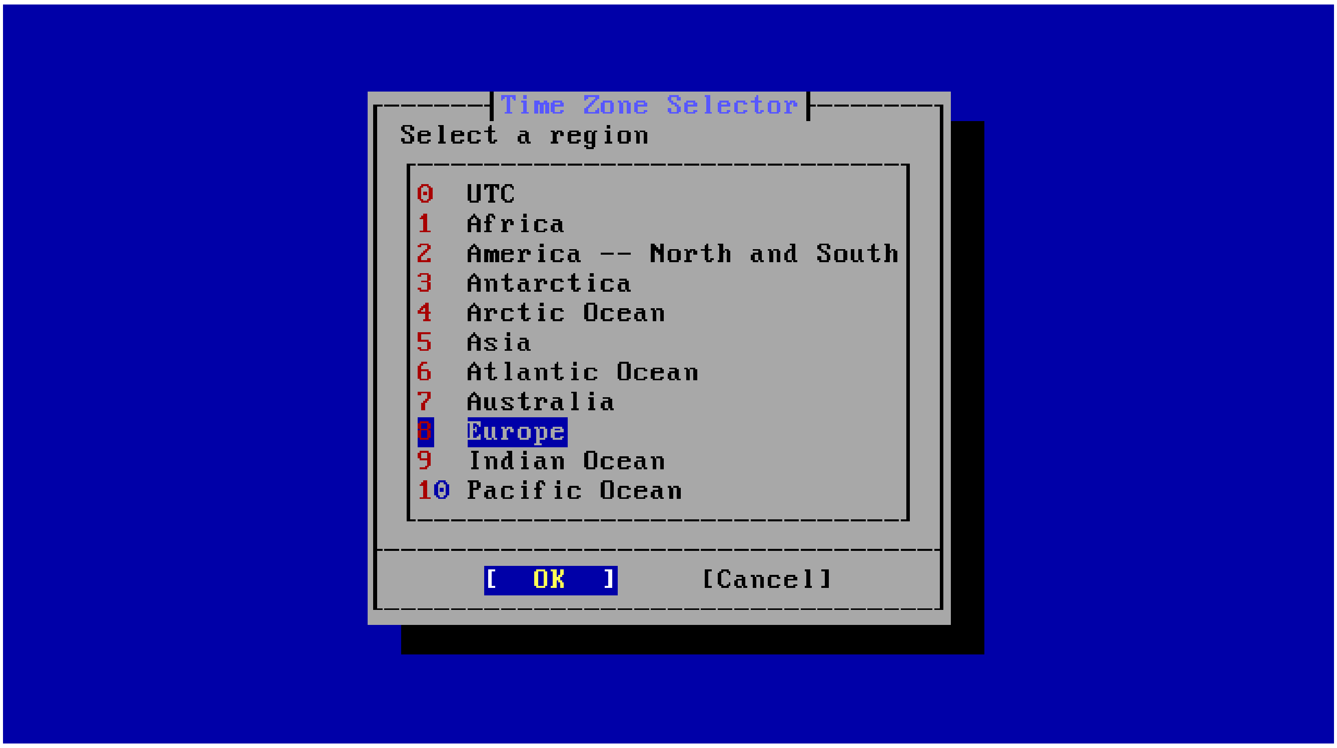
```
FreeBSD Installer
                         Mirror Selection
 Please select the best suitable site for you or "other" if you want to
 specify a different choice. The "Main Site" directs users to the nearest
 project managed mirror via GeoDNS (they carry the full range of possible
 distributions and support both IPv4 and IPv6). All other sites are known
 as "Community Mirrors"; not every site listed here carries more than the
 base distribution kits. Select a site!

           http://ftp.freebsd.org     Main Site (GeoDNS, HTTP)
           ftp://ftp.freebsd.org      Main Site (GeoDNS, FTP)
           http://ftp.au.freebsd.org  Australia - IPv6
           ftp://ftp3.au.freebsd.org  Australia #3
           ftp://ftp.at.freebsd.org   Austria - IPv6
           ftp://ftp2.br.freebsd.org  Brazil #2
           ftp://ftp3.br.freebsd.org  Brazil #3
           ftp://ftp.bg.freebsd.org   Bulgaria - IPv6
           ftp://ftp.cz.freebsd.org   Czech Republic - IPv6
  ↓↓↓                                                          19%

                 [  OK  ]     [Other ]     [Cancel]
```

Figure 46. Choosing a Mirror

2.8.3. Setting the Time Zone

The next series of menus are used to determine the correct local time by selecting the geographic region, country, and time zone. Setting the time zone allows the system to automatically correct for regional time changes, such as daylight savings time, and perform other time zone related functions properly.

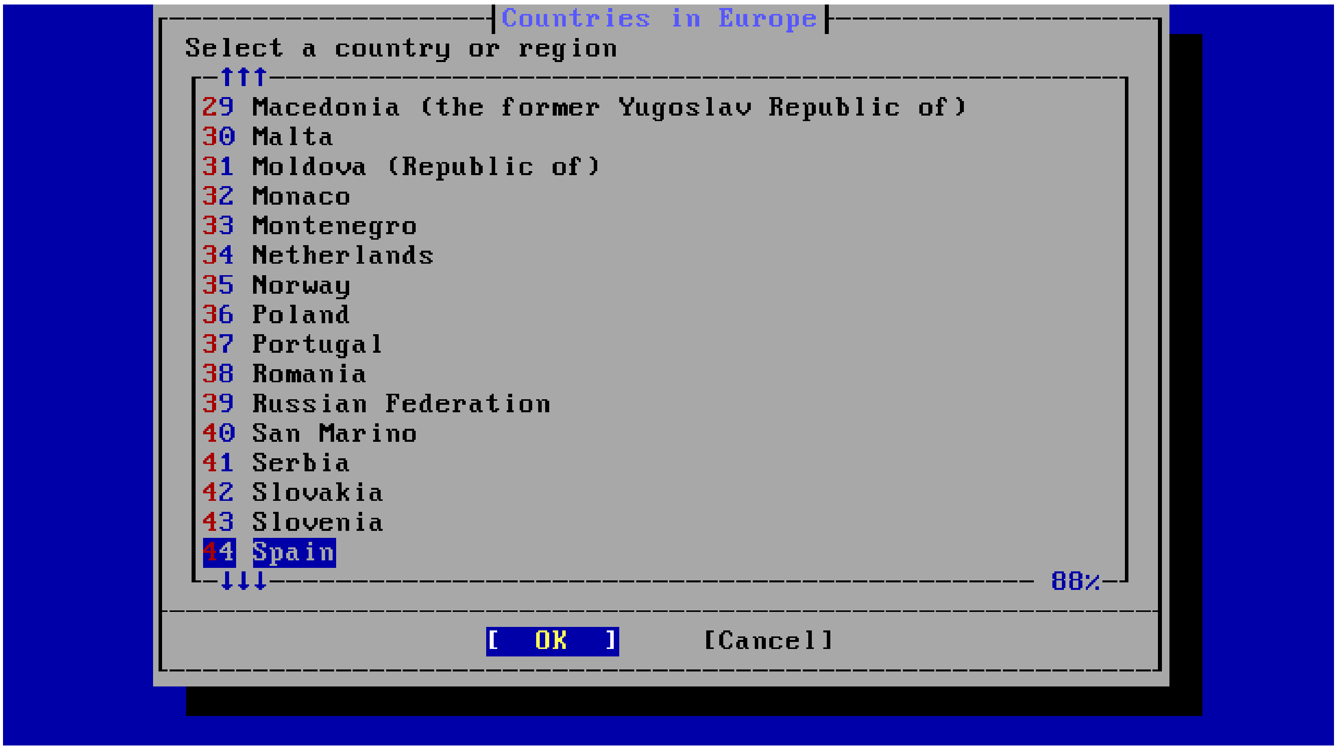
The example shown here is for a machine located in the mainland time zone of Spain, Europe. The selections will vary according to the geographical location.



```
       Time Zone Selector
 Select a region

  0  UTC
  1  Africa
  2  America -- North and South
  3  Antarctica
  4  Arctic Ocean
  5  Asia
  6  Atlantic Ocean
  7  Australia
  8  Europe
  9  Indian Ocean
  10 Pacific Ocean

       [  OK  ]     [Cancel]
```

Figure 47. Select a Region

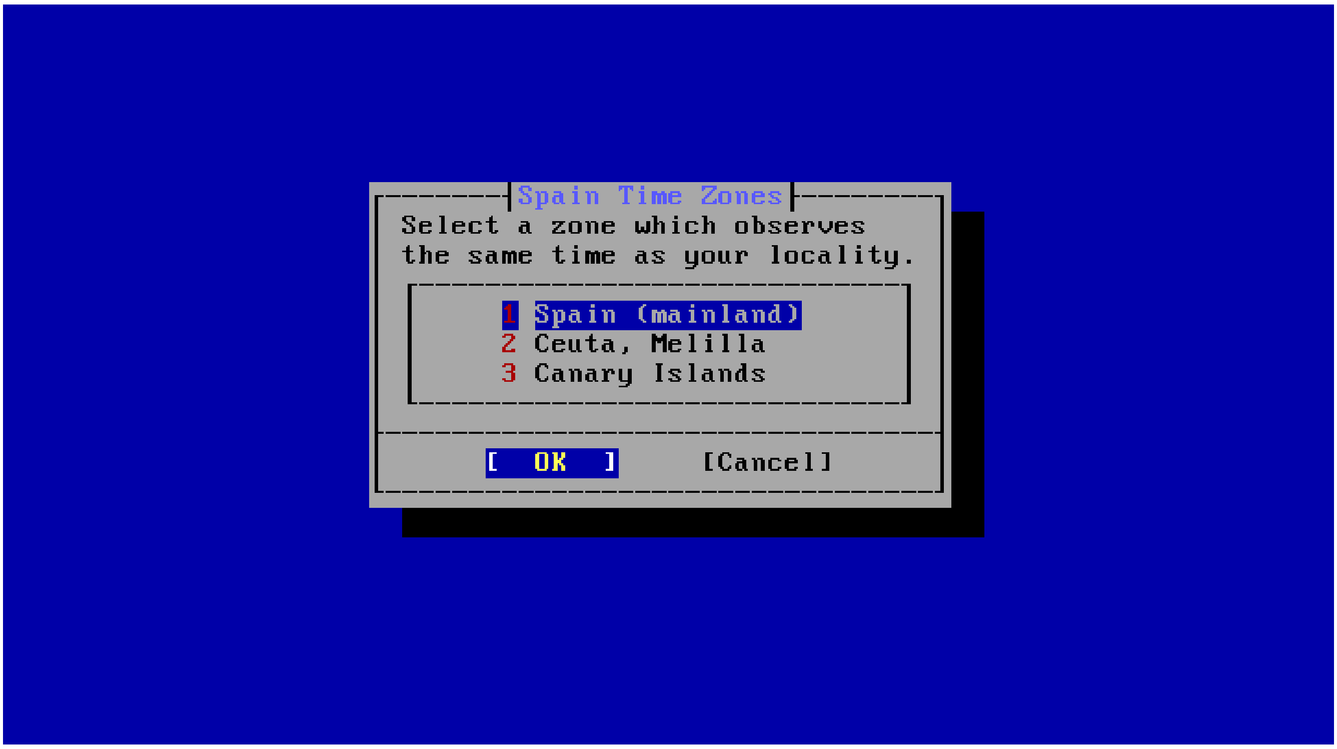
The appropriate region is selected using the arrow keys and then pressing `Enter`.



```
Countries in Europe
Select a country or region
↑↑↑
29 Macedonia (the former Yugoslav Republic of)
30 Malta
31 Moldova (Republic of)
32 Monaco
33 Montenegro
34 Netherlands
35 Norway
36 Poland
37 Portugal
38 Romania
39 Russian Federation
40 San Marino
41 Serbia
42 Slovakia
43 Slovenia
44 Spain
↓↓↓                                       88%

        [  OK  ]          [Cancel]
```

Figure 48. Select a Country

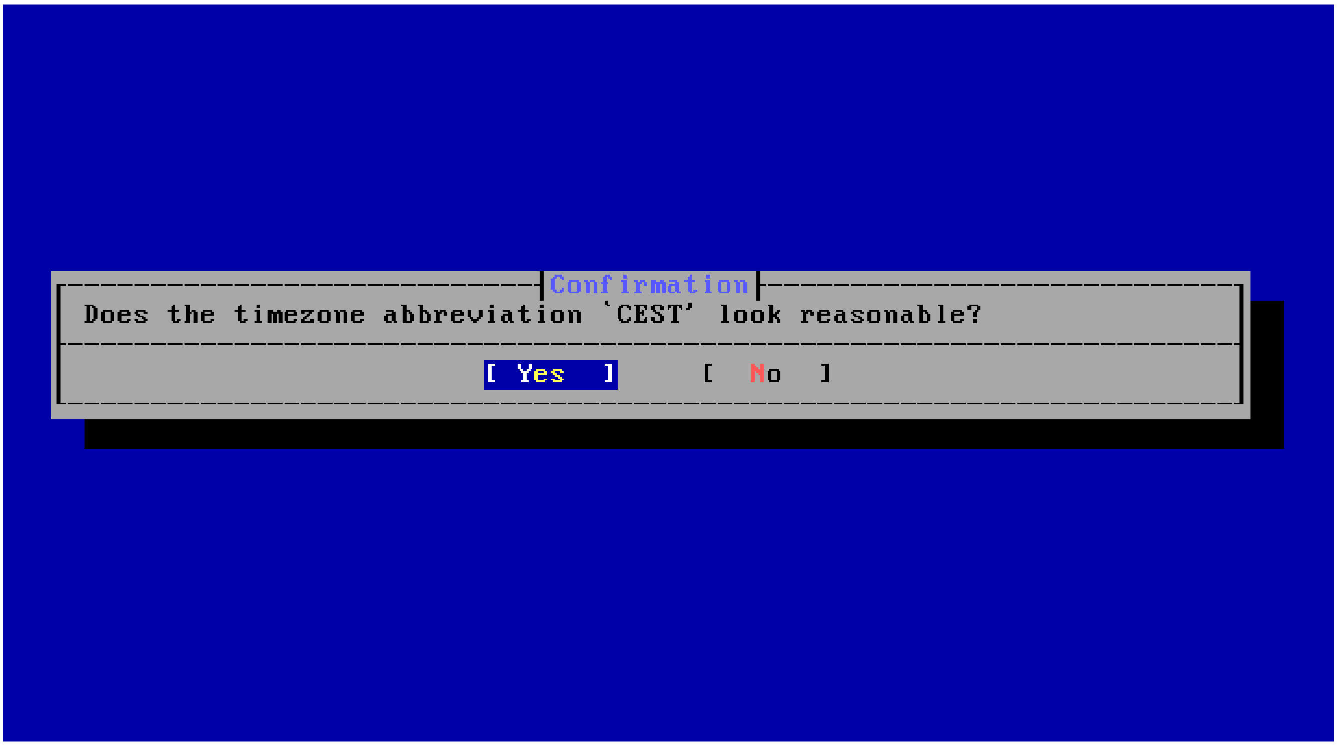
Select the appropriate country using the arrow keys and press `Enter`.



```
Spain Time Zones
Select a zone which observes
the same time as your locality.

      1 Spain (mainland)
      2 Ceuta, Melilla
      3 Canary Islands

  [  OK  ]          [Cancel]
```

Figure 49. Select a Time Zone

The appropriate time zone is selected using the arrow keys and pressing `Enter`.

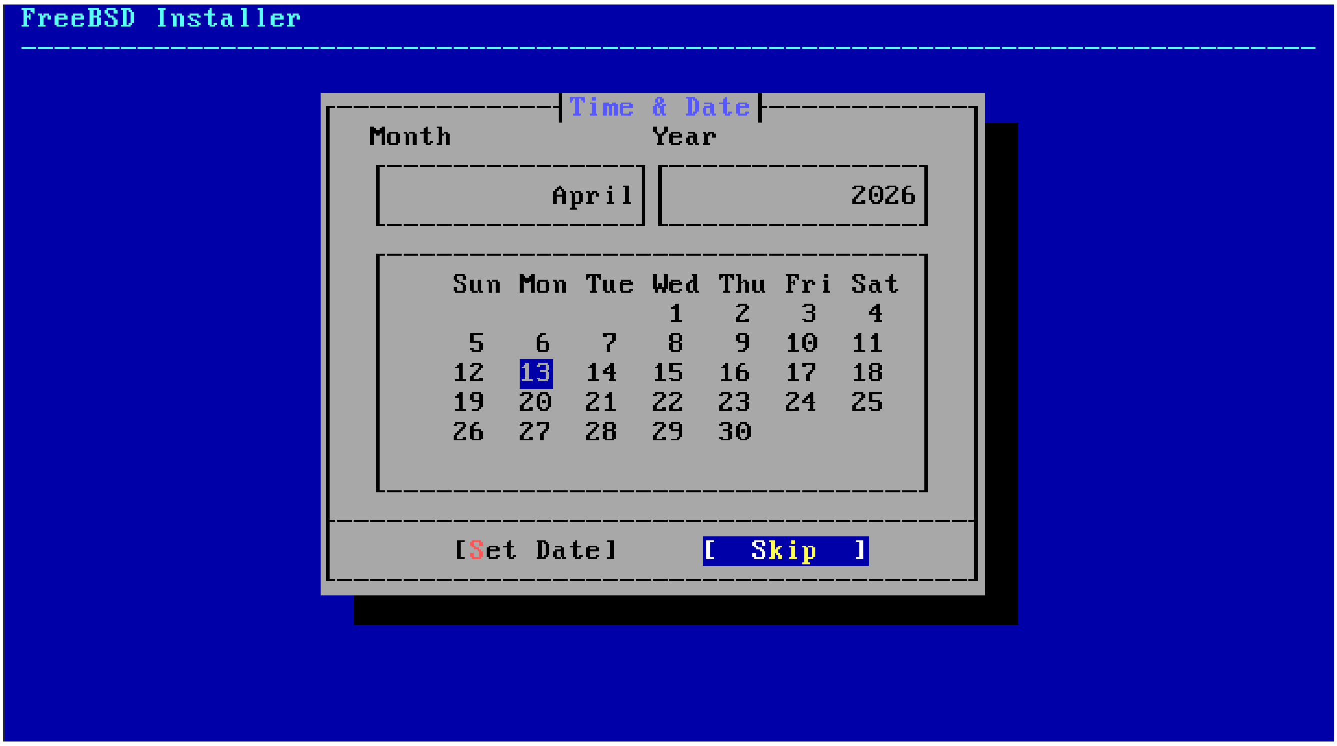

Confirmation

Does the timezone abbreviation `CEST' look reasonable?

[Yes]
[No]

Figure 50. Confirm Time Zone

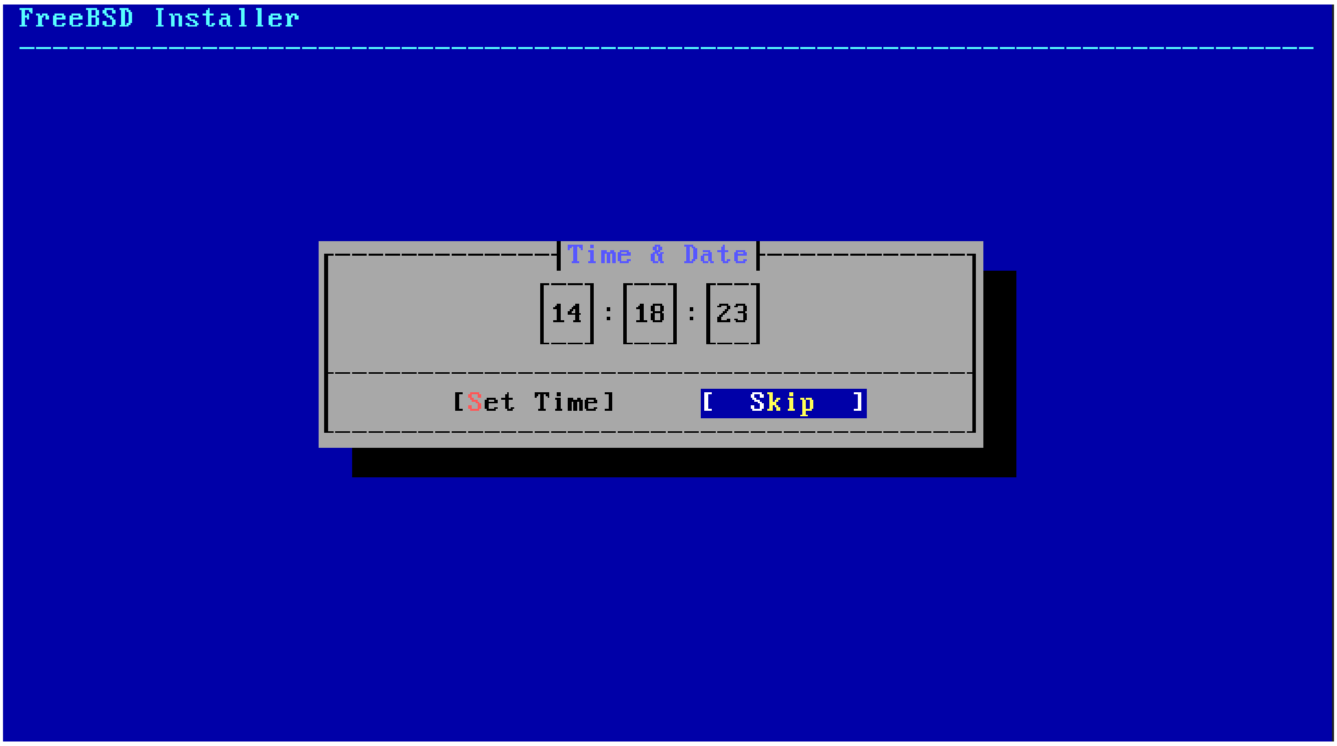
Confirm the abbreviation for the time zone is correct.


FreeBSD Installer
Time & Date
[Set Date]
[Skip]

```
Month                Year
              April                  2026

  Sun Mon Tue Wed Thu Fri Sat
                1   2   3   4
    5   6   7   8   9  10  11
   12  13  14  15  16  17  18
   19  20  21  22  23  24  25
   26  27  28  29  30
```

Figure 51. Select Date

The appropriate date is selected using the arrow keys and then pressing **[Set Date]**. Otherwise, the date selection can be skipped by pressing **[Skip]**.



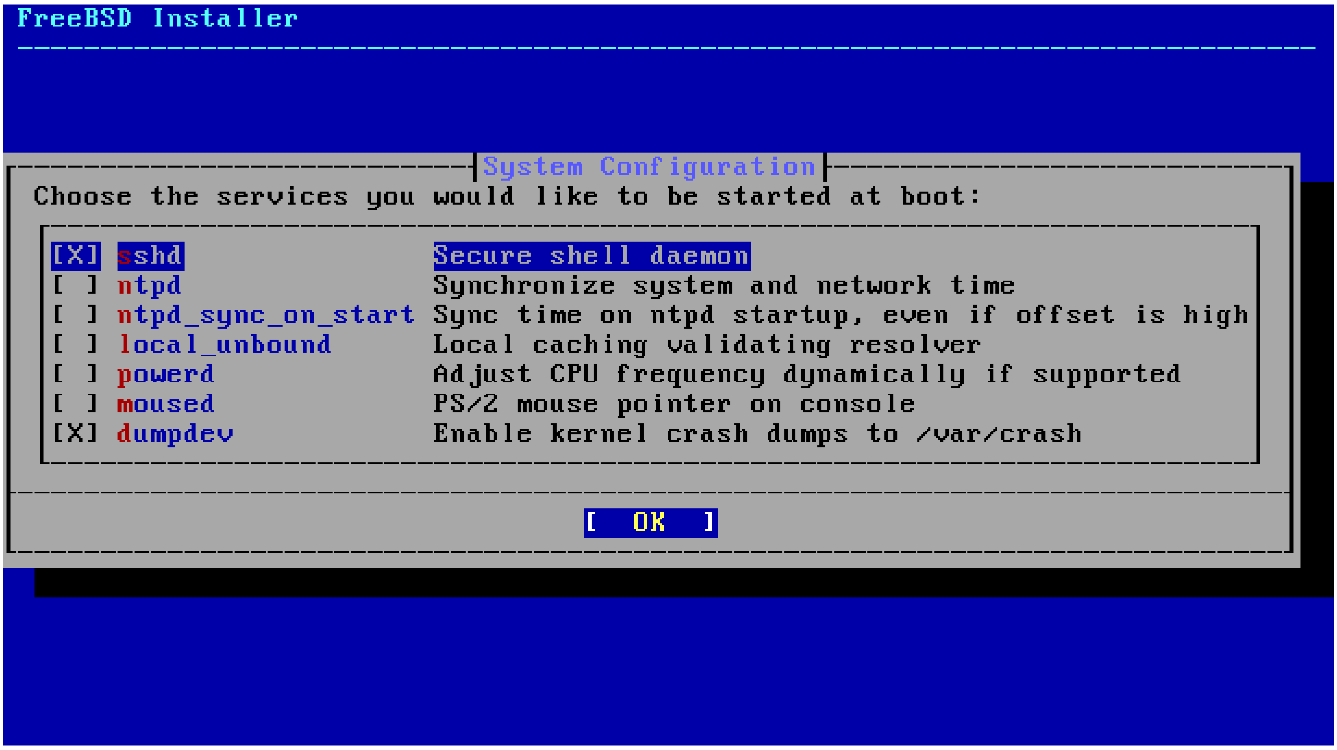
14 : 18 : 23

Figure 52. Select Time

The appropriate time is selected using the arrow keys and then pressing **[Set Time]**. Otherwise, the time selection can be skipped by pressing **[Skip]**.

2.8.4. Enabling Services

The next menu is used to configure which system services will be started whenever the system boots. All of these services are optional. Only start the services that are needed for the system to function.



Choose the services you would like to be started at boot:

```
[X] sshd                Secure shell daemon
[ ] ntpd                Synchronize system and network time
[ ] ntpd_sync_on_start  Sync time on ntpd startup, even if offset is high
[ ] local_unbound       Local caching validating resolver
[ ] powerd              Adjust CPU frequency dynamically if supported
[ ] moused              PS/2 mouse pointer on console
[X] dumpdev             Enable kernel crash dumps to /var/crash
```

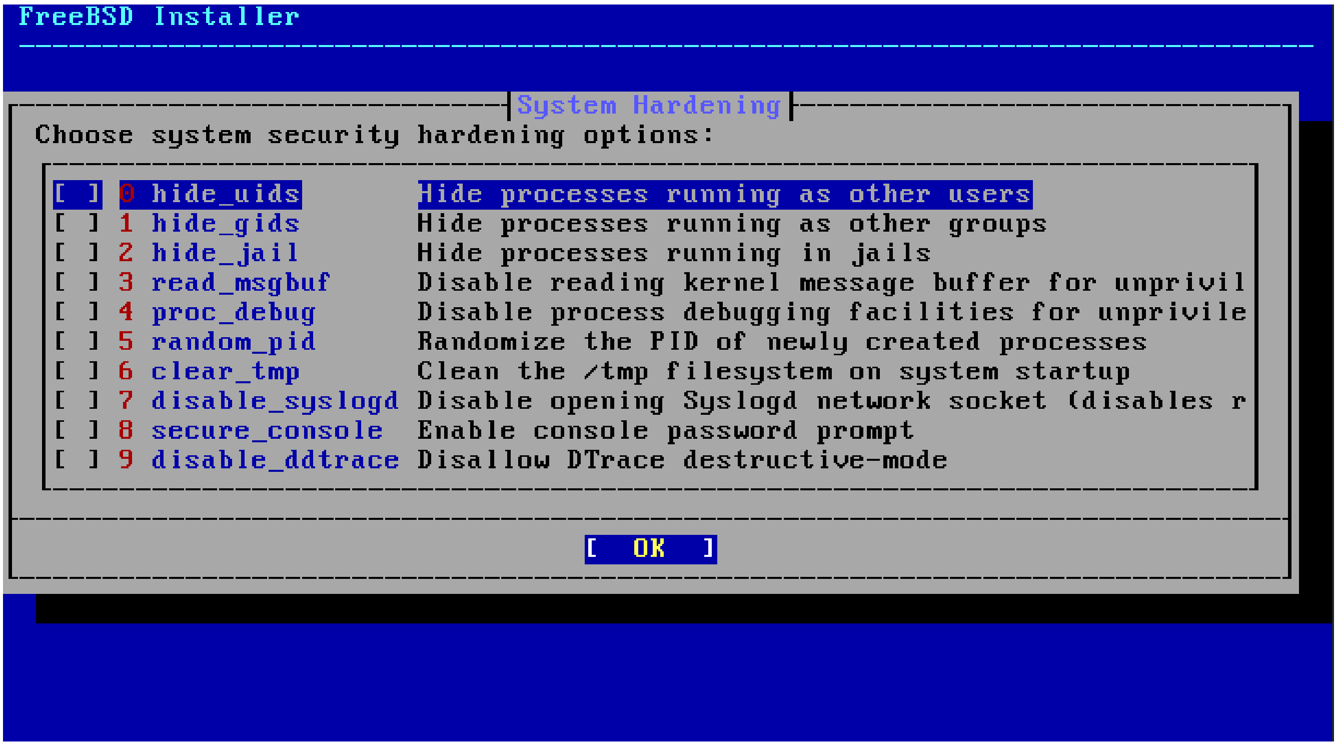
Figure 53. Selecting Additional Services to Enable

Here is a summary of the services that can be enabled in this menu:

- `sshd` - The Secure Shell (SSH) daemon is used to remotely access a system over an encrypted connection. Only enable this service if the system should be available for remote logins.
- `ntpd` - The Network Time Protocol (NTP) daemon for automatic clock synchronization. Enable this service when wanting to synchronise the system clock with a remote time server or pool.
- `ntpd_sync_on_start` - Enabled, ntpd(8) is invoked with the `-g` flag during system startup, performing an immediate, one-time clock synchronization at boot time.
- `local_unbound` - Enable the DNS local unbound. It is necessary to keep in mind that this is a configuration only meant for use as a local caching forwarding resolver. If the objective is to set up a resolver for the entire network, install dns/unbound.
- `powerd` - System power control utility for power control and energy saving.
- `moused` - Enable this service if the mouse will be used from the command-line system console.
- `dumpdev` - Crash dumps are useful when debugging issues with the system, so users are encouraged to enable them.

2.8.5. Enabling Hardening Security Options

The next menu is used to configure which security options will be enabled. All of these options are optional. But their use is encouraged.



```
FreeBSD Installer
------------------------------------------------------------------------------

 ------------------------------| System Hardening |------------------------------
 | Choose system security hardening options:                                    |
 |  ------------------------------------------------------------------------   |
 |  |[ ] 0 hide_uids         Hide processes running as other users         |   |
 |  |[ ] 1 hide_gids         Hide processes running as other groups        |   |
 |  |[ ] 2 hide_jail         Hide processes running in jails               |   |
 |  |[ ] 3 read_msgbuf       Disable reading kernel message buffer for unprivil|
 |  |[ ] 4 proc_debug        Disable process debugging facilities for unprivile|
 |  |[ ] 5 random_pid        Randomize the PID of newly created processes  |   |
 |  |[ ] 6 clear_tmp         Clean the /tmp filesystem on system startup   |   |
 |  |[ ] 7 disable_syslogd Disable opening Syslogd network socket (disables r|
 |  |[ ] 8 secure_console  Enable console password prompt                 |   |
 |  |[ ] 9 disable_ddtrace Disallow DTrace destructive-mode               |   |
 |  ------------------------------------------------------------------------   |
 |------------------------------------------------------------------------------|
 |                                  [  OK  ]                                    |
 --------------------------------------------------------------------------------
```

Figure 54. Selecting Hardening Security Options

Here is a summary of the options that can be enabled in this menu:

- `hide_uids` - Hide processes running as other users (UID). This prevents unprivileged users from seeing running processes from other users.
- `hide_gids` - Hide processes running as other groups (GID). This prevents unprivileged users from

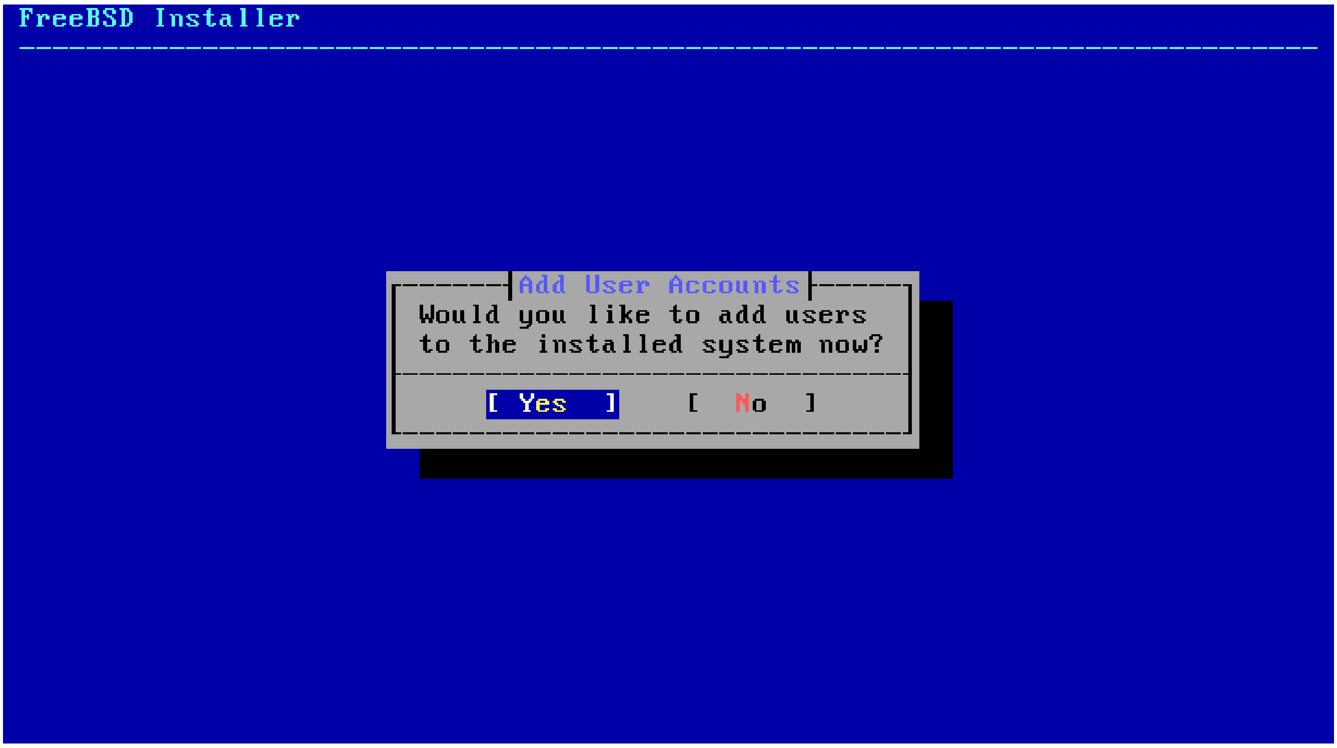
seeing running processes from other groups.

- `hide_jail` - Hide processes running in jails. This prevents unprivileged users from seeing processes running inside jails.
- `read_msgbuf` - Disable reading kernel message buffer for unprivileged users. Prevent unprivileged users from using dmesg(8) to view messages from the kernel's log buffer.
- `proc_debug` - Disable process debugging facilities for unprivileged users. Disables a variety of unprivileged inter-process debugging services, including some procfs functionality, `ptrace()`, and `ktrace()`. Please note that this will also prevent debugging tools such as lldb(1), truss(1) and procstat(1), as well as some built-in debugging facilities in certain scripting languages like PHP.
- `random_pid` - Randomize the PID of processes.
- `clear_tmp` - Clean `/tmp` when the system starts up.
- `disable_syslogd` - Disable opening the syslogd network socket. By default, FreeBSD runs syslogd in a secure way with `-s`. This prevents the daemon from listening for incoming UDP requests on port 514. With this option enabled, syslogd will instead run with `-ss`, which prevents syslogd from opening any port. For more information, see syslogd(8).
- `secure_console` - Make the command prompt request the `root` password when entering single-user mode.
- `disable_ddtrace` - DTrace can run in a mode that affects the running kernel. Destructive actions may not be used unless explicitly enabled. Use `-w` to enable this option when using DTrace. For more information, see dtrace(1).

2.8.6. Add Users

The next menu prompts to create at least one user account. It is recommended to log into the system using a user account rather than as `root`. When logged in as `root`, there are essentially no limits or protection on what can be done. Logging in as a normal user is safer and more secure.

Select **[Yes]** to add new users.


FreeBSD Installer

Add User Accounts

Would you like to add users to the installed system now?

Figure 55. Add User Accounts

Follow the prompts and input the requested information for the user account. The example shown in Enter User Information creates the `asample` user account.

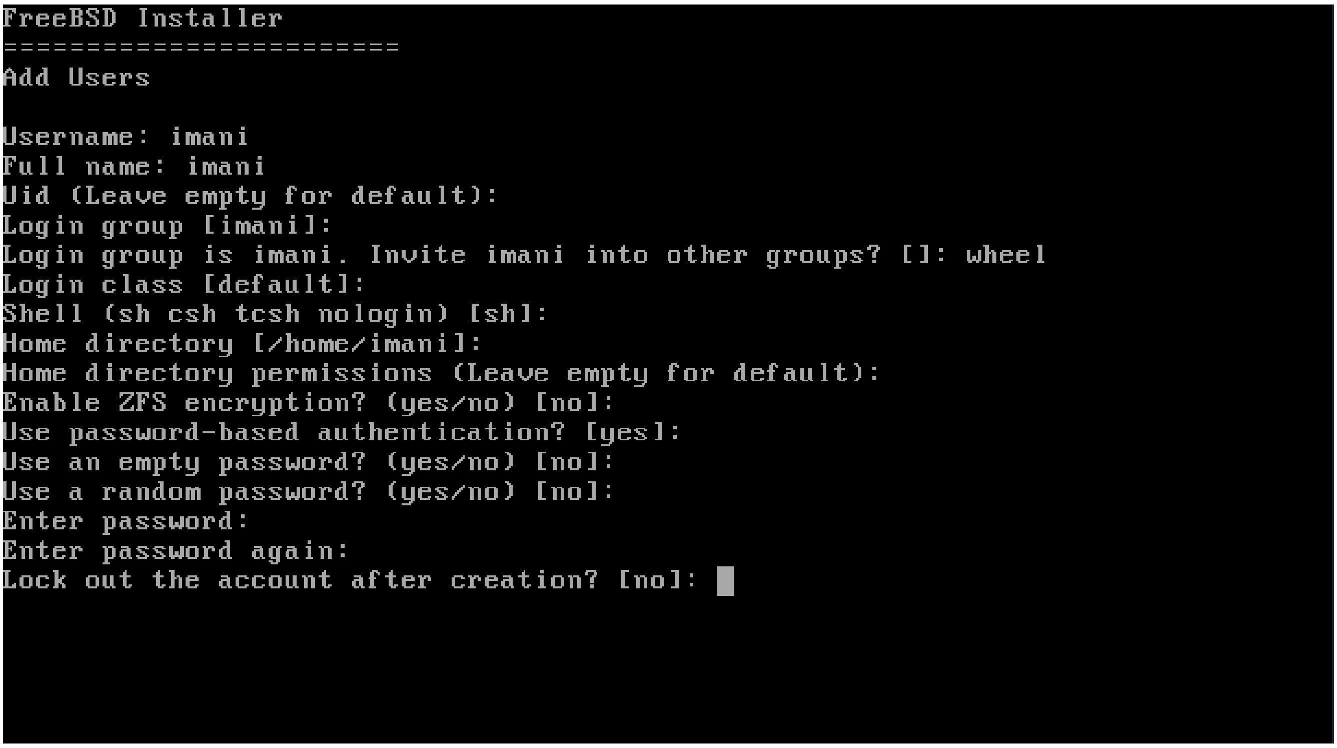

```
FreeBSD Installer
========================
Add Users

Username: imani
Full name: imani
Uid (Leave empty for default):
Login group [imani]:
Login group is imani. Invite imani into other groups? []: wheel
Login class [default]:
Shell (sh csh tcsh nologin) [sh]:
Home directory [/home/imani]:
Home directory permissions (Leave empty for default):
Enable ZFS encryption? (yes/no) [no]:
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
```

Figure 56. Enter User Information

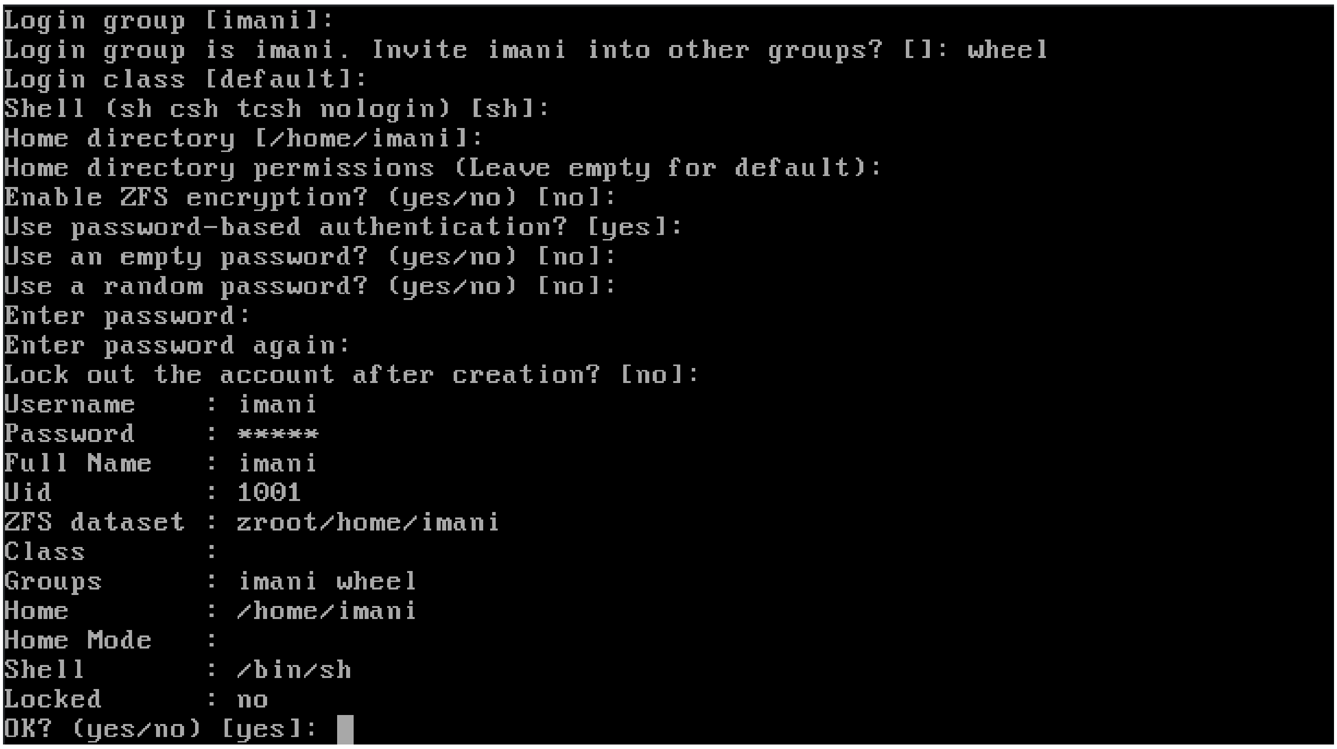
Here is a summary of the information to input:

- `Username` - The name the user will enter to log in. A common convention is to use the first letter of the first name combined with the last name, as long as each username is unique for the system. The username is case sensitive and should not contain any spaces.
- `Full name` - The user’s full name. This can contain spaces and is used as a description for the

user account.

- `Uid` - User ID. This is typically left blank so the system automatically assigns a value.
- `Login group` - The user's group. This is typically left blank to accept the default.
- `Invite user into other groups?` - Additional groups to which the user will be added as a member. If the user needs administrative access, type `wheel` here.
- `Login class` - Typically left blank for the default.
- `Shell` - Type in one of the listed values to set the interactive shell for the user. Refer to Shells for more information about shells.
- `Home directory` - The user's home directory. The default is usually correct.
- `Home directory permissions` - Permissions on the user's home directory. The default is usually correct.
- `Use password-based authentication?` - Typically `yes` so that the user is prompted to input their password at login.
- `Use an empty password?` - Typically `no` as empty or blank passwords are insecure.
- `Use a random password?` - Typically `no` so that the user can set their own password in the next prompt.
- `Enter password` - The password for this user. Typed-in characters will not be shown on the screen.
- `Enter password again` - The password must be typed again for verification.
- `Lock out the account after creation?` - Typically `no` so that the user can log in.

After entering all the details, a summary is shown for review. If a mistake was made, enter `no` to correct it. Once everything is correct, enter `yes` to create the new user.



```
Login group [imani]:
Login group is imani. Invite imani into other groups? []: wheel
Login class [default]:
Shell (sh csh tcsh nologin) [sh]:
Home directory [/home/imani]:
Home directory permissions (Leave empty for default):
Enable ZFS encryption? (yes/no) [no]:
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username    : imani
Password    : *****
Full Name   : imani
Uid         : 1001
ZFS dataset : zroot/home/imani
Class       :
Groups      : imani wheel
Home        : /home/imani
Home Mode   :
Shell       : /bin/sh
Locked      : no
OK? (yes/no) [yes]:
```

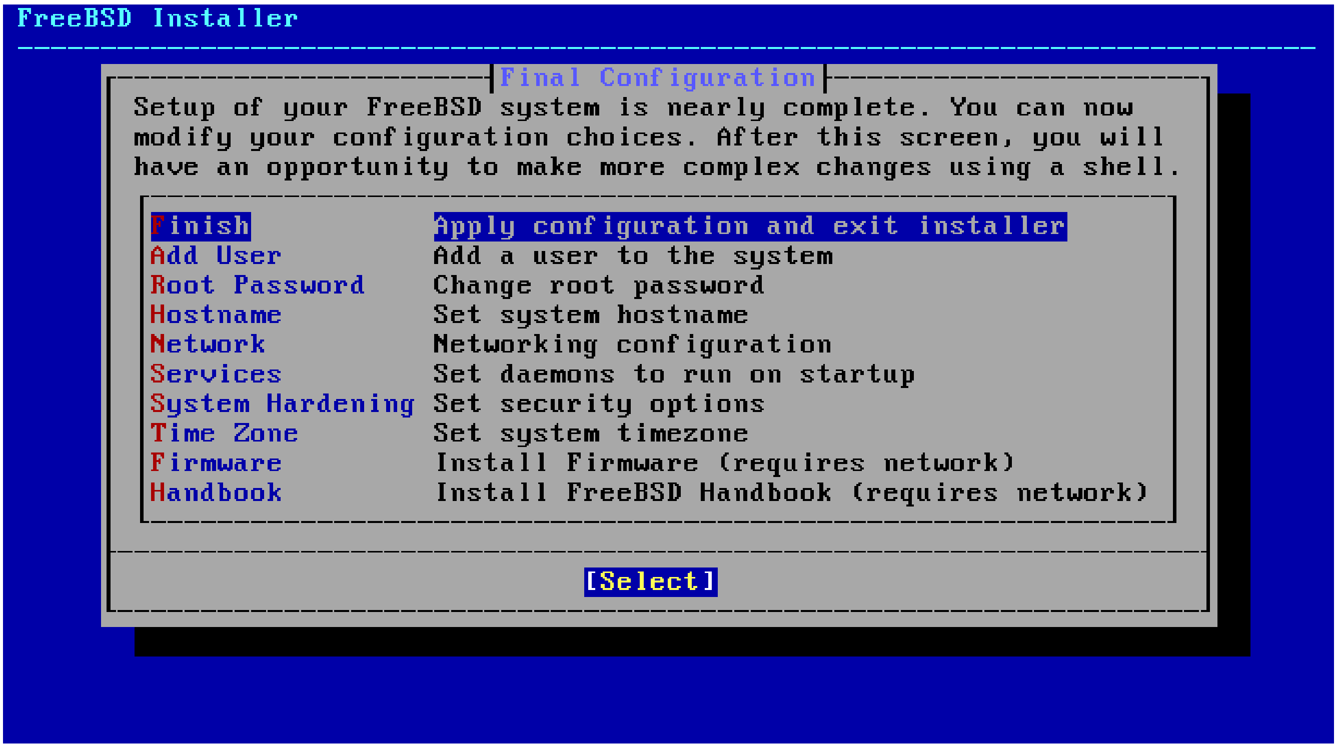
Figure 57. Exit User and Group Management

If there are more users to add, answer the `Add another user?` question with `yes`. Enter `no` to finish adding users and continue the installation.

For more information on adding users and user management, see Users and Basic Account Management.

2.8.7. Final Configuration

After everything has been installed and configured, a final chance is provided to modify settings.


FreeBSD Installer
Final Configuration
Finish	Apply configuration and exit installer
Add User	Add a user to the system
Root Password	Change root password
Hostname	Set system hostname
Network	Networking configuration
Services	Set daemons to run on startup
System Hardening	Set security options
Time Zone	Set system timezone
Firmware	Install Firmware (requires network)
Handbook	Install FreeBSD Handbook (requires network)
[Select]

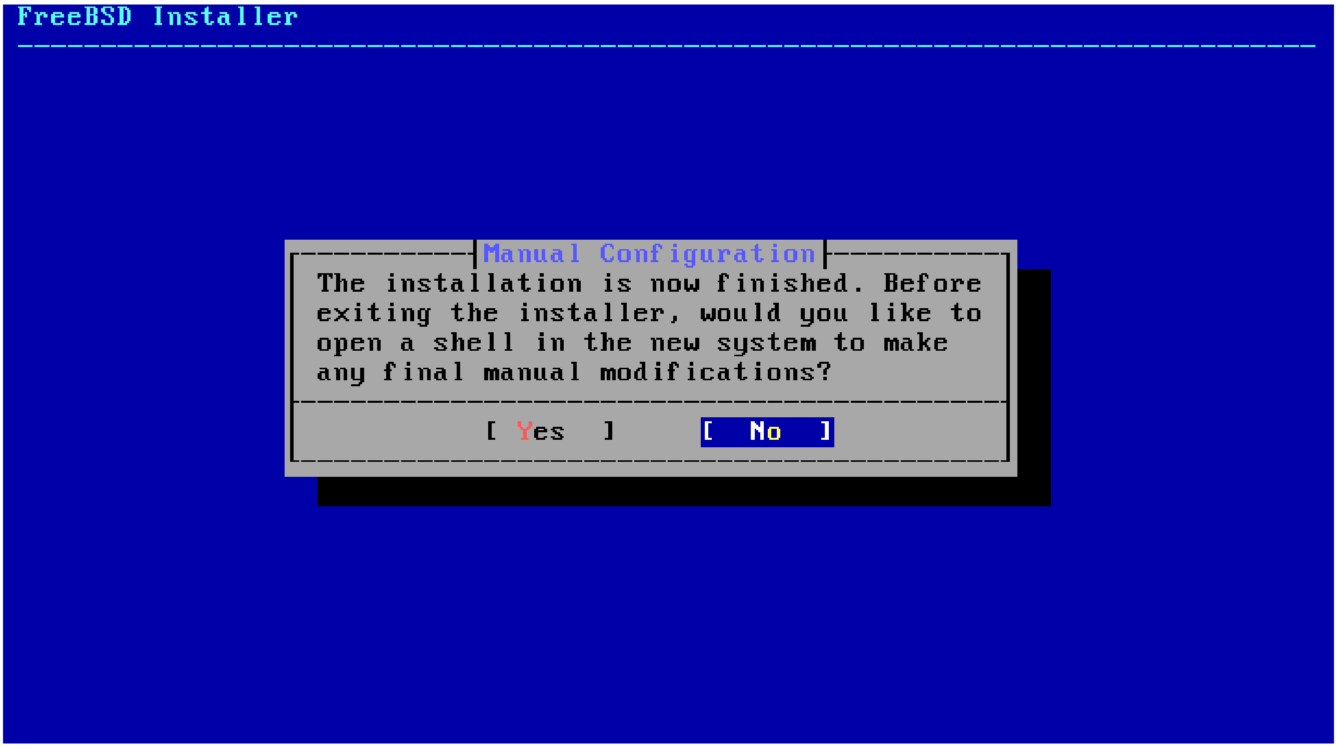
Setup of your FreeBSD system is nearly complete. You can now modify your configuration choices. After this screen, you will have an opportunity to make more complex changes using a shell.

Figure 58. Final Configuration

Use this menu to make any changes or to do any additional configuration before completing the installation.

- `Add User` - Described in Add Users.
- `Root Password` - Described in Setting the `root` Password.
- `Hostname` - Described in Setting the Hostname.
- `Network` - Described in Configuring Network Interfaces.
- `Services` - Described in Enabling Services.
- `System Hardening` - Described in Enabling Hardening Security Options.
- `Time Zone` - Described in Setting the Time Zone.
- `Firmware` - Download and install required firmware for system components such as Wi-Fi or graphics cards.
- `Handbook` - Download and install the FreeBSD Handbook.

Once configuration is complete, select **[Exit]**.

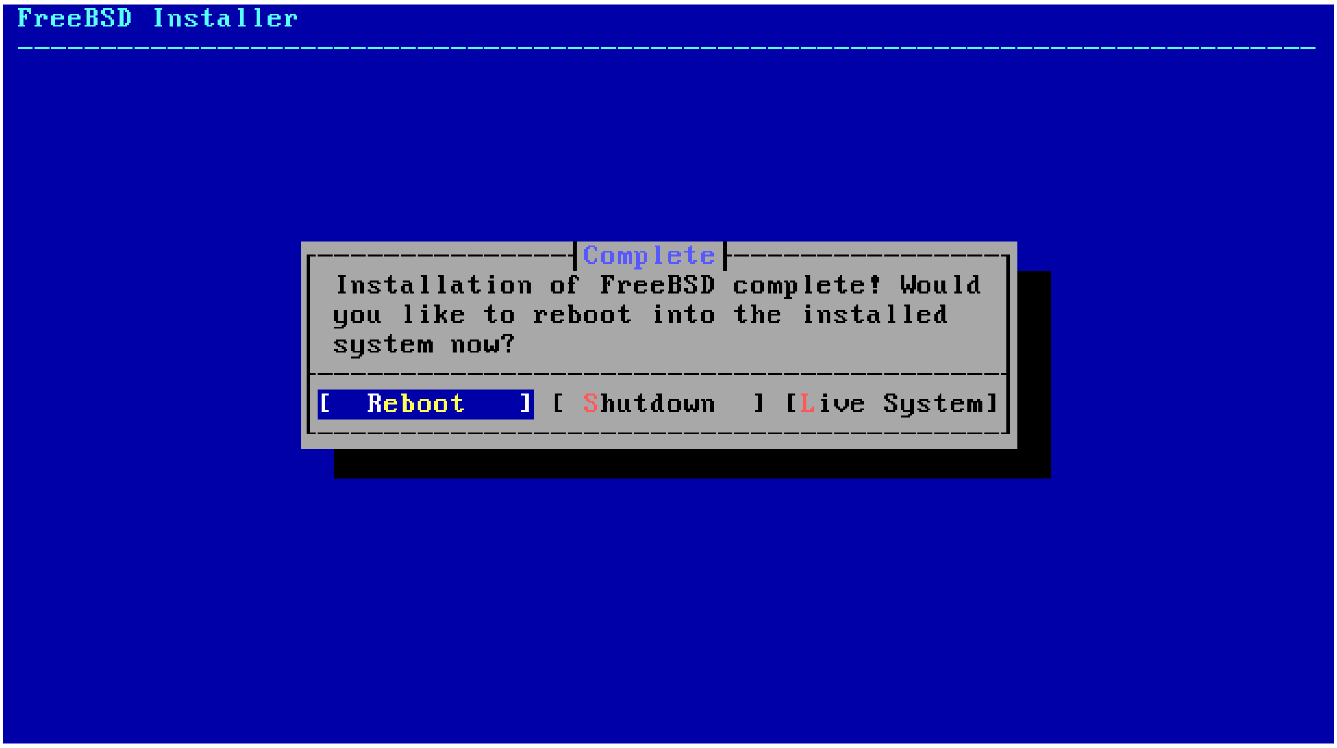

FreeBSD Installer

Manual Configuration

The installation is now finished. Before exiting the installer, would you like to open a shell in the new system to make any final manual modifications?

Figure 59. Manual Configuration

bsdinstall will prompt for any additional configuration that needs to be done before rebooting into the new system. Select **[Yes]** to exit to a shell within the new system or **[No]** to proceed to the last step of the installation.


FreeBSD Installer

Complete

Installation of FreeBSD complete! Would you like to reboot into the installed system now?

Figure 60. Complete the Installation

If further configuration or special setup is needed, select **[Live CD]** to boot the install media into Live CD mode.

If the installation is complete, select **[Reboot]** to reboot the computer and start the new FreeBSD system. Do not forget to remove the FreeBSD install media or the computer might boot from it again.

As FreeBSD boots, informational messages are displayed. After the system finishes booting, a login prompt is displayed. At the `login:` prompt, enter the username added during the installation. Avoid logging in as `root`. Refer to The Superuser Account for instructions on how to become the superuser when administrative access is needed.

The messages that appear during boot can be reviewed by pressing `Scroll-Lock` to turn on the scroll-back buffer. The `PgUp`, `PgDn`, and arrow keys can be used to scroll back through the messages. When finished, press `Scroll-Lock` again to unlock the display and return to the console. To review these messages once the system has been up for some time, type `less /var/run/dmesg.boot` from a command prompt. Press `q` to return to the command line after viewing.

If sshd was enabled in Selecting Additional Services to Enable, the first boot might be a bit slower as the system generates SSH host keys. Subsequent boots will be faster. The fingerprints of the keys are then displayed as in the following example:

```
Generating public/private rsa1 key pair.
Your identification has been saved in /etc/ssh/ssh_host_key.
Your public key has been saved in /etc/ssh/ssh_host_key.pub.
The key fingerprint is:
10:a0:f5:af:93:ae:a3:1a:b2:bb:3c:35:d9:5a:b3:f3 root@machine3.example.com
The key's randomart image is:
+--[RSA1 1024]----+
|    o..          |
|   o . .         |
|  .   o          |
|       o         |
|    o   S        |
|   + + o         |
|o . + *          |
|o+ ..+ .         |
|==o..o+E         |
+-----------------+
Generating public/private dsa key pair.
Your identification has been saved in /etc/ssh/ssh_host_dsa_key.
Your public key has been saved in /etc/ssh/ssh_host_dsa_key.pub.
The key fingerprint is:
7e:1c:ce:dc:8a:3a:18:13:5b:34:b5:cf:d9:d1:47:b2 root@machine3.example.com
The key's randomart image is:
+--[ DSA 1024]----+
|       ..     . .|
|      o  .   . + |
|     . ..   . E .|
|    . .  o o . . |
|     +  S = .    |
|    +  . = o     |
|     +  . * .    |
|    . .  o .     |
|      .o. .      |
+-----------------+
Starting sshd.
```

Refer to OpenSSH for more information about fingerprints and SSH.

FreeBSD does not install a graphical environment by default. Refer to The X Window System for more information about installing and configuring a graphical window manager.

Proper shutdown of a FreeBSD computer helps protect data and hardware from damage. *Do not turn off the power before the system has been properly shut down!* If the user is a member of the `wheel` group, become the superuser by typing `su` at the command line and entering the `root` password. Then, type `shutdown -p now` and the system will shut down cleanly, and, if the hardware supports it, turn itself off.

2.9. Troubleshooting

This section covers basic installation troubleshooting, such as common problems people have reported.

Check the Hardware Notes listed on the FreeBSD Release Information page for the version of FreeBSD to make sure the hardware is supported.

> Some installation problems can be avoided or alleviated by updating the firmware on various hardware components, most notably the motherboard. Motherboard firmware is usually referred to as the BIOS. Most motherboard and computer manufacturers have a website for upgrades and upgrade information.
>
> Manufacturers generally advise against upgrading the motherboard BIOS unless there is a good reason for doing so, like a critical update. The upgrade process *can* go wrong, leaving the BIOS incomplete and the computer inoperative.

If the system hangs while probing hardware during boot or behaves strangely during the installation process, ACPI may be the culprit. FreeBSD makes extensive use of the system ACPI service on the i386 and amd64 platforms to aid in system configuration if it is detected during boot. Unfortunately, some bugs still exist in both the ACPI driver and within system motherboards and BIOS firmware. ACPI can be disabled by setting the `hint.acpi.0.disabled` hint in the third stage boot loader:

```
set hint.acpi.0.disabled="1"
```

This is reset each time the system is booted, so it is necessary to add `hint.acpi.0.disabled="1"` to the file `/boot/loader.conf`. More information about the boot loader can be found in "Synopsis".

2.10. Using the Live CD

The welcome menu of bsdinstall, shown in Welcome Menu, provides a **[Live CD]** option. This is useful for those who are still wondering whether FreeBSD is the right operating system for them and want to test some of the features before installing.

The following points should be noted before using the **[Live CD]**:

- To gain access to the system, authentication is required. The username is `root` and the password is blank.
- As the system runs directly from the installation media, performance will be significantly slower than that of a system installed on a hard disk.
- This option only provides a command prompt and not a graphical interface.

Chapter 3. FreeBSD Basics

3.1. Synopsis

This chapter covers the basic commands and functionality of the FreeBSD operating system. Much of this material is relevant for any UNIX®-like operating system. New FreeBSD users are encouraged to read through this chapter carefully.

Read this chapter to learn:

- How to use and configure virtual consoles.
- How to create and manage users and groups on FreeBSD.
- How UNIX® file permissions and FreeBSD file flags work.
- The default FreeBSD file system layout.
- The FreeBSD disk organization.
- How to mount and unmount file systems.
- What processes, daemons, and signals are.
- What a shell is, and how to change the default login environment.
- How to use basic text editors.
- What devices and device nodes are.
- How to read manual pages for more information.

3.2. Virtual Consoles and Terminals

Unless FreeBSD has been configured to automatically start a graphical environment during startup, the system will boot into a command line login prompt, as seen in this example:

```
FreeBSD/amd64 (pc3.example.org) (ttyv0)

login:
```

The first line contains some information about the system. The `amd64` indicates that FreeBSD is running on a 64-bit x86 system. The hostname is `pc3.example.org`, and `ttyv0` indicates that this is the "system console". The second line is the login prompt.

Since FreeBSD is a multiuser system, it needs some way to distinguish between different users. This is accomplished by requiring every user to log into the system before gaining access to the programs on the system. Every user has a unique "username" and a personal "password".

To log into the system console, type the username that was configured during system installation, as described in Add Users, and press `Enter`. Then enter the password associated with the username and press `Enter`. The password is *not echoed* for security reasons.

Once the correct password is input, the message of the day (MOTD) will be displayed followed by a command prompt. Depending upon the shell that was selected when the user was created, this prompt will be a `#`, `$`, or `%` character. The prompt indicates that the user is now logged into the FreeBSD system console and ready to try the available commands.

3.2.1. Virtual Consoles

While the system console can be used to interact with the system, a user working from the command line at the keyboard of a FreeBSD system will typically instead log into a virtual console. This is because system messages are configured by default to display on the system console. These messages will appear over the command or file that the user is working on, making it difficult to concentrate on the work at hand.

By default, FreeBSD is configured to provide several virtual consoles for inputting commands. Each virtual console has its own login prompt and shell and it is easy to switch between virtual consoles. This essentially provides the command line equivalent of having several windows open at the same time in a graphical environment.

The key combinations `Alt`+`F1` through `Alt`+`F8` have been reserved by FreeBSD for switching between virtual consoles. Use `Alt`+`F1` to switch to the system console (`ttyv0`), `Alt`+`F2` to access the first virtual console (`ttyv1`), `Alt`+`F3` to access the second virtual console (`ttyv2`), and so on. When using Xorg as a graphical console, the combination becomes `Ctrl`+`Alt`+`F1` to return to a text-based virtual console.

When switching from one console to the next, FreeBSD manages the screen output. The result is an illusion of having multiple virtual screens and keyboards that can be used to type commands for FreeBSD to run. The programs that are launched in one virtual console do not stop running when the user switches to a different virtual console.

Refer to kbdcontrol(1), vidcontrol(1), atkbd(4), syscons(4), and vt(4) for a more technical description of the FreeBSD console and its keyboard drivers.

In FreeBSD, the number of available virtual consoles is configured in this section of `/etc/ttys`:

```
# name    getty                         type  status comments
#
ttyv0   "/usr/libexec/getty Pc"         xterm   on  secure
# Virtual terminals
ttyv1   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv2   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv3   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv4   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv5   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv6   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv7   "/usr/libexec/getty Pc"         xterm   on  secure
ttyv8   "/usr/X11R6/bin/xdm -nodaemon"  xterm   off secure
```

To disable a virtual console, put a comment symbol (`#`) at the beginning of the line representing that virtual console. For example, to reduce the number of available virtual consoles from eight to four, put a `#` in front of the last four lines representing virtual consoles `ttyv5` through `ttyv8`. *Do not* comment out the line for the system console `ttyv0`. Note that the last virtual console (`ttyv8`) is used to access the graphical environment if Xorg has been installed and configured as described in The X Window System.

For a detailed description of every column in this file and the available options for the virtual consoles, refer to ttys(5).

3.2.2. Single User Mode

The FreeBSD boot menu provides an option labelled as "Boot Single User". If this option is selected, the system will boot into a special mode known as "single user mode". This mode is typically used to repair a system that will not boot or to reset the `root` password when it is not known. While in single user mode, networking and other virtual consoles are not available. However, full `root` access to the system is available, and by default, the `root` password is not needed. For these reasons, physical access to the keyboard is needed to boot into this mode and determining who has physical access to the keyboard is something to consider when securing a FreeBSD system.

The settings which control single user mode are found in this section of `/etc/ttys`:

```
# name  getty                           type  status  comments
#
# If console is marked "insecure", then init will ask for the root password
# when going to single-user mode.
console none                            unknown  off  secure
```

By default, the status is set to `secure`. This assumes that who has physical access to the keyboard is either not important or it is controlled by a physical security policy. If this setting is changed to `insecure`, the assumption is that the environment itself is insecure because anyone can access the keyboard. When this line is changed to `insecure`, FreeBSD will prompt for the `root` password when a user selects to boot into single user mode.

Be careful when changing this setting to `insecure`*!* If the `root` password is forgotten, booting into single user mode is still possible, but may be difficult for someone who is not familiar with the FreeBSD booting process.

3.2.3. Changing Console Video Modes

The FreeBSD console default video mode may be adjusted to 1024x768, 1280x1024, or any other size supported by the graphics chip and monitor. To use a different video mode load the `VESA` module:

```
# kldload vesa
```

To determine which video modes are supported by the hardware, use vidcontrol(1). To get a list of supported video modes issue the following:

```
# vidcontrol -i mode
```

The output of this command lists the video modes that are supported by the hardware. To select a new video mode, specify the mode using vidcontrol(1) as the `root` user:

```
# vidcontrol MODE_279
```

If the new video mode is acceptable, it can be permanently set on boot by adding it to `/etc/rc.conf`:

```
allscreens_flags="MODE_279"
```

3.3. Users and Basic Account Management

FreeBSD allows multiple users to use the computer at the same time. While only one user can sit in front of the screen and use the keyboard at any one time, any number of users can log in to the system through the network. To use the system, each user should have their own user account.

This chapter describes:

- The different types of user accounts on a FreeBSD system.
- How to add, remove, and modify user accounts.
- How to set limits to control the resources that users and groups are allowed to access.
- How to create groups and add users as members of a group.

3.3.1. Account Types

Since all access to the FreeBSD system is achieved using accounts and all processes are run by users, user and account management is important.

There are three main types of accounts: system accounts, user accounts, and the superuser account.

System Accounts

System accounts are used to run services such as DNS, mail, and web servers. The reason for this is security; if all services ran as the superuser, they could act without restriction.

Examples of system accounts are `daemon`, `operator`, `bind`, `news`, and `www`.

`nobody` is the generic unprivileged system account. However, the more services that use `nobody`, the more files and processes that user will become associated with, and hence the more privileged that user becomes.

User Accounts

User accounts are assigned to real people and are used to log in and use the system. Every person accessing the system should have a unique user account. This allows the administrator to find out who is doing what and prevents users from clobbering the settings of other users.

Each user can set up their own environment to accommodate their use of the system, by configuring their default shell, editor, key bindings, and language settings.

Every user account on a FreeBSD system has certain information associated with it:

User name

The user name is typed at the `login:` prompt. Each user must have a unique user name. There are a number of rules for creating valid user names which are documented in passwd(5). It is recommended to use user names that consist of eight or fewer, all lower case characters in order to maintain backwards compatibility with applications.

Password

Each account has an associated password.

User ID (UID)

The User ID (UID) is a number used to uniquely identify the user to the FreeBSD system. Commands that allow a user name to be specified will first convert it to the UID. It is recommended to use a UID less than 65535, since higher values may cause compatibility issues with some software.

Group ID (GID)

The Group ID (GID) is a number used to uniquely identify the primary group that the user belongs to. Groups are a mechanism for controlling access to resources based on a user's GID rather than their UID. This can significantly reduce the size of some configuration files and allows users to be members of more than one group. It is recommended to use a GID of 65535 or lower as higher GIDs may break some software.

Login class

Login classes are an extension to the group mechanism that provide additional flexibility when tailoring the system to different users. Login classes are discussed further in Configuring Login Classes.

Password change time

By default, passwords do not expire. However, password expiration can be enabled on a per-user basis, forcing some or all users to change their passwords after a certain amount of time has elapsed.

Account expiration time

By default, FreeBSD does not expire accounts. When creating accounts that need a limited lifespan, such as student accounts in a school, specify the account expiry date using pw(8). After the expiry time has elapsed, the account cannot be used to log in to the system, although the account's directories and files will remain.

User's full name

The user name uniquely identifies the account to FreeBSD, but does not necessarily reflect the user's real name. Similar to a comment, this information can contain spaces, uppercase characters, and be more than 8 characters long.

Home directory

The home directory is the full path to a directory on the system. This is the user's starting directory when the user logs in. A common convention is to put all user home directories under `/home/username` or `/usr/home/username`. Each user stores their personal files and subdirectories in their own home directory.

User shell

The shell provides the user's default environment for interacting with the system. There are many different kinds of shells and experienced users will have their own preferences, which can be reflected in their account settings.

The Superuser Account

The superuser account, usually called `root`, is used to manage the system with no limitations on privileges. For this reason, it should not be used for day-to-day tasks like sending and receiving mail, general exploration of the system, or programming.

The superuser, unlike other user accounts, can operate without limits, and misuse of the superuser account may result in spectacular disasters. User accounts are unable to destroy the operating system by mistake, so it is recommended to login as a user account and to only become the superuser when a command requires extra privilege.

Always double and triple-check any commands issued as the superuser, since an extra space or missing character can mean irreparable data loss.

There are several ways to gain superuser privilege. While one can log in as `root`, this is highly discouraged.

Instead, use su(1) to become the superuser. If `-` is specified when running this command, the user will also inherit the root user's environment. The user running this command must be in the `wheel` group or else the command will fail. The user must also know the password for the `root` user account.

In this example, the user only becomes superuser in order to run `make install` as this step requires superuser privilege. Once the command completes, the user types `exit` to leave the superuser account and return to the privilege of their user account.

Example 2. Install a Program As the Superuser

```
% configure
% make
% su -
Password:
# make install
# exit
%
```

The built-in su(1) framework works well for single systems or small networks with just one system administrator. An alternative is to install the security/sudo package or port. This software provides activity logging and allows the administrator to configure which users can run which commands as the superuser.

3.3.2. Managing Accounts

FreeBSD provides a variety of different commands to manage user accounts. The most common commands are summarized in Utilities for Managing User Accounts, followed by some examples of their usage. See the manual page for each utility for more details and usage examples.

Table 2. Utilities for Managing User Accounts

Command	Summary
adduser(8)	The recommended command-line application for adding new users.
rmuser(8)	The recommended command-line application for removing users.
chpass(1)	A flexible tool for changing user database information.
passwd(1)	The command-line tool to change user passwords.
pw(8)	A powerful and flexible tool for modifying all aspects of user accounts.
bsdconfig(8)	A system configuration utility with account management support.

Adding a user

The recommended program for adding new users is adduser(8). When a new user is added, this program automatically updates `/etc/passwd` and `/etc/group`. It also creates a home directory for the new user, copies in the default configuration files from `/usr/share/skel`, and can optionally mail the new user a welcome message. This utility must be run as the superuser.

The adduser(8) utility is interactive and walks through the steps for creating a new user account. As seen in Adding a User on FreeBSD, either input the required information or press `Return` to accept the default value shown in square brackets. In this example, the user has been invited into the `wheel` group, allowing them to become the superuser with su(1). When finished, the utility will prompt to either create another user or to exit.

Example 3. Adding a User on FreeBSD

```
# adduser
```

The output should be similar to the following:

```
Username: jru
Full name: J. Random User
Uid (Leave empty for default):
Login group [jru]:
Login group is jru. Invite jru into other groups? []: wheel
Login class [default]:
```

```
Shell (sh csh tcsh zsh nologin) [sh]: zsh
Home directory [/home/jru]:
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]:
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username   : jru
Password   : ****
Full Name  : J. Random User
Uid        : 1001
Class      :
Groups     : jru wheel
Home       : /home/jru
Shell      : /usr/local/bin/zsh
Locked     : no
OK? (yes/no): yes
adduser: INFO: Successfully added (jru) to the user database.
Add another user? (yes/no): no
Goodbye!
```

Since the password is not echoed when typed, be careful to not mistype the password when creating the user account.

Removing a user

To completely remove a user from the system, run rmuser(8) as the superuser. This command performs the following steps:

1. Removes the user's crontab(1) entry, if one exists.
2. Removes any at(1) jobs belonging to the user.
3. Sends a SIGKILL signal to all processes owned by the user.
4. Removes the user from the system's local password file.
5. Removes the user's home directory (if it is owned by the user), including handling of symbolic links in the path to the actual home directory.
6. Removes the incoming mail files belonging to the user from `/var/mail`.
7. Removes all files owned by the user from `/tmp`, `/var/tmp`, and `/var/tmp/vi.recover`.
8. Removes the username from all groups to which it belongs in `/etc/group`. (If a group becomes empty and the group name is the same as the username, the group is removed; this complements adduser(8)'s per-user unique groups.)
9. Removes all message queues, shared memory segments and semaphores owned by the user.

rmuser(8) cannot be used to remove superuser accounts since that is almost always an indication of massive destruction.

By default, an interactive mode is used, as shown in the following example.

Example 4. `rmuser` Interactive Account Removal

```
# rmuser jru
```

The output should be similar to the following:

```
Matching password entry:
jru:*:1001:1001::0:0:J. Random User:/home/jru:/usr/local/bin/zsh
Is this the entry you wish to remove? y
Remove user's home directory (/home/jru)? y
Removing user (jru): mailspool home passwd.
```

Change user information

Any user can use chpass(1) to change their default shell and personal information associated with their user account. The superuser can use this utility to change additional account information for any user.

When passed no options, aside from an optional username, chpass(1) displays an editor containing user information. When the user exits from the editor, the user database is updated with the new information.

> This utility will prompt for the user's password when exiting the editor, unless the utility is run as the superuser.

In Using `chpass` as Superuser, the superuser has typed `chpass jru` and is now viewing the fields that can be changed for this user. If `jru` runs this command instead, only the last six fields will be displayed and available for editing. This is shown in Using `chpass` as Regular User.

Example 5. Using `chpass` as Superuser

```
# chpass jru
```

The output should be similar to the following:

```
# Changing user database information for jru.
Login: jru
Password: *
Uid [#]: 1001
Gid [# or name]: 1001
```

```
Change [month day year]:
Expire [month day year]:
Class:
Home directory: /home/jru
Shell: /usr/local/bin/zsh
Full Name: J. Random User
Office Location:
Office Phone:
Home Phone:
Other information:
```

Example 6. Using `chpass` as Regular User

```
#Changing user database information for jru.
Shell: /usr/local/bin/zsh
Full Name: J. Random User
Office Location:
Office Phone:
Home Phone:
Other information:
```

> The commands chfn(1) and chsh(1) are links to chpass(1), as are ypchpass(1), ypchfn(1), and ypchsh(1). Since NIS support is automatic, specifying the `yp` before the command is not necessary. How to configure NIS is covered in Network Servers.

Change user password

Any user can easily change their password using passwd(1). To prevent accidental or unauthorized changes, this command will prompt for the user's original password before a new password can be set:

Example 7. Changing The Password

```
% passwd
```

The output should be similar to the following:

```
Changing local password for jru.
Old password:
New password:
Retype new password:
passwd: updating the database...
passwd: done
```

The superuser can change any user's password by specifying the username when running passwd(1). When this utility is run as the superuser, it will not prompt for the user's current password. This allows the password to be changed when a user cannot remember the original password.

Example 8. Changing Another User's Password as the Superuser

```
# passwd jru
```

The output should be similar to the following:

```
Changing local password for jru.
New password:
Retype new password:
passwd: updating the database...
passwd: done
```

As with chpass(1), yppasswd(1) is a link to passwd(1), so NIS works with either command.

Create, remove, modify and display system users and groups

The pw(8) utility can create, remove, modify, and display users and groups. It functions as a front end to the system user and group files. pw(8) has a very powerful set of command line options that make it suitable for use in shell scripts, but new users may find it more complicated than the other commands presented in this section.

3.3.3. Managing Groups

A group is a list of users. A group is identified by its group name and GID. In FreeBSD, the kernel uses the UID of a process, and the list of groups it belongs to, to determine what the process is allowed to do. Most of the time, the GID of a user or process usually means the first group in the list.

The group name to GID mapping is listed in `/etc/group`. This is a plain text file with four colon-delimited fields. The first field is the group name, the second is the encrypted password, the third the GID, and the fourth the comma-delimited list of members. For a complete description of the syntax, refer to group(5).

The superuser can modify `/etc/group` using a text editor, although editing the group file using vigr(8) is preferred because it can catch some common mistakes. Alternatively, pw(8) can be used to add and edit groups. For example, to add a group called `teamtwo` and then confirm that it exists:

Care must be taken when using the operator group, as unintended superuser-like access privileges may be granted, including but not limited to shutdown, reboot, and access to all items in `/dev` in the group.

Example 9. Adding a Group Using pw(8)

```
# pw groupadd teamtwo
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:
```

In this example, `1100` is the GID of `teamtwo`. Right now, `teamtwo` has no members. This command will add `jru` as a member of `teamtwo`.

Example 10. Adding User Accounts to a New Group Using pw(8)

```
# pw groupmod teamtwo -M jru
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:jru
```

The argument to `-M` is a comma-delimited list of users to be added to a new (empty) group or to replace the members of an existing group. To the user, this group membership is different from (and in addition to) the user's primary group listed in the password file. This means that the user will not show up as a member when using `groupshow` with pw(8), but will show up when the information is queried via id(1) or a similar tool. When pw(8) is used to add a user to a group, it only manipulates `/etc/group` and does not attempt to read additional data from `/etc/passwd`.

Example 11. Adding a New Member to a Group Using pw(8)

```
# pw groupmod teamtwo -m db
# pw groupshow teamtwo
```

The output should be similar to the following:

```
teamtwo:*:1100:jru,db
```

In this example, the argument to `-m` is a comma-delimited list of users who are to be added to the group. Unlike the previous example, these users are appended to the group and do not replace existing users in the group.

Example 12. Using id(1) to Determine Group Membership

```
% id jru
```

The output should be similar to the following:

```
uid=1001(jru) gid=1001(jru) groups=1001(jru), 1100(teamtwo)
```

In this example, `jru` is a member of the groups `jru` and `teamtwo`.

For more information about this command and the format of `/etc/group`, refer to pw(8) and group(5).

3.4. Permissions

In FreeBSD, every file and directory has an associated set of permissions and several utilities are available for viewing and modifying these permissions. Understanding how permissions work is necessary to make sure that users are able to access the files that they need and are unable to improperly access the files used by the operating system or owned by other users.

This section discusses the traditional UNIX® permissions used in FreeBSD. For finer-grained file system access control, refer to Access Control Lists.

In UNIX®, basic permissions are assigned using three types of access: read, write, and execute. These access types are used to determine file access to the file's owner, group, and others (everyone else). The read, write, and execute permissions can be represented as the letters `r`, `w`, and `x`. They can also be represented as binary numbers as each permission is either on or off (`0`). When represented as a number, the order is always read as `rwx`, where `r` has an on value of `4`, `w` has an on value of `2` and `x` has an on value of `1`.

Table 4.1 summarizes the possible numeric and alphabetic possibilities. When reading the "Directory Listing" column, a `-` is used to represent a permission that is set to off.

Table 3. UNIX® Permissions

Value	Permission	Directory Listing
0	No read, no write, no execute	`---`
1	No read, no write, execute	`--x`
2	No read, write, no execute	`-w-`
3	No read, write, execute	`-wx`
4	Read, no write, no execute	`r--`
5	Read, no write, execute	`r-x`
6	Read, write, no execute	`rw-`

Value	Permission	Directory Listing
7	Read, write, execute	`rwx`

Use the `-l` argument with ls(1) to view a long directory listing that includes a column of information about a file's permissions for the owner, group, and everyone else. For example, `ls -l` in an arbitrary directory may show:

```
% ls -l
```

The output should be similar to the following:

```
total 530
-rw-r--r--  1 root  wheel     512 Sep  5 12:31 myfile
-rw-r--r--  1 root  wheel     512 Sep  5 12:31 otherfile
-rw-r--r--  1 root  wheel    7680 Sep  5 12:31 email.txt
```

Focusing on the line for `myfile`, the first `(leftmost)` character indicates whether this file is a regular file, a directory, a special character device, a socket, or any other special pseudo-file device. In this example, the `-` indicates a regular file. The next three characters, `rw-` in this example, give the permissions for the owner of the file. The next three characters, `r--`, give the permissions for the group that the file belongs to. The final three characters, `r--`, give the permissions for the rest of the world. A dash means that the permission is turned off. In this example, the permissions are set so the owner can read and write to the file, the group can read the file, and the rest of the world can only read the file. According to the table above, the permissions for this file would be `644`, where each digit represents the three parts of the file's permission.

How does the system control permissions on devices? FreeBSD treats most hardware devices as a file that programs can open, read, and write data to. These special device files are stored in `/dev/`.

Directories are also treated as files. They have read, write, and execute permissions. The executable bit for a directory has a slightly different meaning than that of files. When a directory is marked executable, it means it is possible to change into that directory using cd(1). This also means that it is possible to access the files within that directory, subject to the permissions on the files themselves.

In order to perform a directory listing, the read permission must be set on the directory. In order to delete a file that one knows the name of, it is necessary to have write *and* execute permissions to the directory containing the file.

There are more permission bits, but they are primarily used in special circumstances such as setuid binaries and sticky directories. For more information on file permissions and how to set them, refer to chmod(1).

3.4.1. Symbolic Permissions

Symbolic permissions use characters instead of octal values to assign permissions to files or directories. Symbolic permissions use the syntax of (who) (action) (permissions), where the following values are available:

Option	Letter	Represents
(who)	u	User
(who)	g	Group owner
(who)	o	Other
(who)	a	All ("world")
(action)	+	Adding permissions
(action)	-	Removing permissions
(action)	=	Explicitly set permissions
(permissions)	r	Read
(permissions)	w	Write
(permissions)	x	Execute
(permissions)	t	Sticky bit
(permissions)	s	Set UID or GID

These values are used with chmod(1), but with letters instead of numbers. For example, the following command would block both members of the group associated with *FILE* and all other users from accessing *FILE*:

```
% chmod go= FILE
```

A comma separated list can be provided when more than one set of changes to a file must be made. For example, the following command removes the group and "world" write permission on *FILE*, and adds the execute permissions for everyone:

```
% chmod go-w,a+x FILE
```

3.4.2. FreeBSD File Flags

In addition to file permissions, FreeBSD supports the use of "file flags". These flags add an additional level of security and control over files, but not directories. With file flags, even `root` can be prevented from removing or altering files.

File flags are modified using chflags(1). For example, to enable the system undeletable flag on the file `file1`, issue the following command:

```
# chflags sunlink file1
```

To disable the system undeletable flag, put a "no" in front of the `sunlink`:

```
# chflags nosunlink file1
```

To view the flags of a file, use `-lo` with ls(1):

```
# ls -lo file1
```

```
-rw-r--r--  1 trhodes  trhodes  sunlnk 0 Mar  1 05:54 file1
```

Several file flags may only be added or removed by the `root` user. In other cases, the file owner may set its file flags. Refer to chflags(1) and chflags(2) for more information.

3.4.3. The setuid, setgid, and sticky Permissions

Other than the permissions already discussed, there are three other specific settings that all administrators should know about. They are the `setuid`, `setgid`, and `sticky` permissions.

These settings are important for some UNIX® operations as they provide functionality not normally granted to normal users. To understand them, the difference between the real user ID and effective user ID must be noted.

The real user ID is the UID who owns or starts the process. The effective UID is the user ID the process runs as. As an example, passwd(1) runs with the real user ID when a user changes their password. However, in order to update the password database, the command runs as the effective ID of the `root` user. This allows users to change their passwords without seeing a `Permission Denied` error.

The setuid permission may be added symbolically by adding the `s` permission for the user as in the following example:

```
# chmod u+s suidexample.sh
```

The setuid permission may also be set by prefixing a permission set with the number four (4) as shown in the following example:

```
# chmod 4755 suidexample.sh
```

The permissions on `suidexample.sh` now look like the following:

```
-rwsr-xr-x   1 trhodes  trhodes    63 Aug 29 06:36 suidexample.sh
```

Note that a `s` is now part of the permission set designated for the file owner, replacing the executable bit. This allows utilities which need elevated permissions, such as passwd(1).

ℹ The `nosuid` mount(8) option will cause such binaries to silently fail without alerting the user. That option is not completely reliable as a `nosuid` wrapper may be able to circumvent it.

To view this in real time, open two terminals. On one, type `passwd` as a normal user. While it waits for a new password, check the process table and look at the user information for passwd(1):

In terminal A:

```
Changing local password for trhodes
Old Password:
```

In terminal B:

```
# ps aux | grep passwd
```

```
trhodes  5232  0.0  0.2  3420  1608   0  R+    2:10AM   0:00.00 grep passwd
root     5211  0.0  0.2  3620  1724   2  I+    2:09AM   0:00.01 passwd
```

Although passwd(1) is run as a normal user, it is using the effective UID of `root`.

The `setgid` permission performs the same function as the `setuid` permission; except that it alters the group settings. When an application or utility executes with this setting, it will be granted the permissions based on the group that owns the file, not the user who started the process.

To set the `setgid` permission on a file symbolically, add the `s` permission for the group with chmod(1):

```
# chmod g+s sgidexample.sh
```

Alternatively, provide chmod(1) with a leading two (2):

```
# chmod 2755 sgidexample.sh
```

In the following listing, notice that the `s` is now in the field designated for the group permission settings:

```
-rwxr-sr-x   1 trhodes  trhodes    44 Aug 31 01:49 sgidexample.sh
```

ℹ In these examples, even though the shell script in question is an executable file, it will not run with a different EUID or effective user ID. This is because shell scripts may not access the setuid(2) system calls.

The `setuid` and `setgid` permission bits may lower system security, by allowing for elevated permissions. The third special permission, the `sticky bit`, can strengthen the security of a system.

When the `sticky bit` is set on a directory, it allows file deletion only by the file owner. This is useful to prevent file deletion in public directories, such as `/tmp`, by users who do not own the file. To utilize this permission, add the `t` mode to the file:

```
# chmod +t /tmp
```

Alternatively, prefix the permission set with a one (1):

```
# chmod 1777 /tmp
```

The `sticky bit` permission will display as a `t` at the very end of the permission set:

```
# ls -al / | grep tmp
```

```
drwxrwxrwt  10 root  wheel         512 Aug 31 01:49 tmp
```

3.5. Directory Structure

The FreeBSD directory hierarchy is fundamental to obtaining an overall understanding of the system. The most important directory is root or, "/". This directory is the first one mounted at boot time and it contains the base system necessary to prepare the operating system for multi-user operation. The root directory also contains mount points for other file systems that are mounted during the transition to multi-user operation.

A mount point is a directory where additional file systems can be grafted onto a parent file system (usually the root file system). This is further described in Disk Organization. Standard mount points include `/usr/`, `/var/`, `/tmp/`, `/mnt/`, and `/media/`. These directories are usually referenced to entries in `/etc/fstab`. This file is a table of various file systems and mount points and is read by the system. Most of the file systems in `/etc/fstab` are mounted automatically at boot time from the script rc(8) unless their entry includes `noauto`. Details can be found in The fstab File.

A complete description of the file system hierarchy is available in hier(7). The following table provides a brief overview of the most common directories.

Directory	Description
`/`	Root directory of the file system.
`/bin/`	User utilities fundamental to both single-user and multi-user environments.
`/boot/`	Programs and configuration files used during operating system bootstrap.

`/boot/defaults/`	Default boot configuration files. Refer to loader.conf(5) for details.
`/dev/`	Device special files managed by devfs(5)
`/etc/`	System configuration files and scripts.
`/etc/defaults/`	Default system configuration files. Refer to rc(8) for details.
`/etc/periodic/`	Scripts that run daily, weekly, and monthly, via cron(8). Refer to periodic(8) for details.
`/lib/`	Critical system libraries needed for binaries in `/bin` and `/sbin`
`/libexec/`	Critical system files
`/media/`	Contains subdirectories to be used as mount points for removable media such as CDs, USB drives, and floppy disks
`/mnt/`	Empty directory commonly used by system administrators as a temporary mount point.
`/net/`	Automounted NFS shares; see auto_master(5)
`/proc/`	Process file system. Refer to procfs(5) for details.
`/rescue/`	Statically linked programs for emergency recovery as described in rescue(8).
`/root/`	Home directory for the `root` account.
`/sbin/`	System programs and administration utilities fundamental to both single-user and multi-user environments.
`/tmp/`	Temporary files which are usually *not* preserved across a system reboot. A memory-based file system is often mounted at `/tmp`. This can be automated using the tmpmfs-related variables of rc.conf(5) or with an entry in `/etc/fstab`; refer to mdmfs(8) for details.
`/usr/`	The majority of user utilities and applications.
`/usr/bin/`	Common utilities, programming tools, and applications.
`/usr/include/`	Standard C include files.
`/usr/lib/`	Archive libraries.
`/usr/libdata/`	Miscellaneous utility data files.
`/usr/libexec/`	System daemons and system utilities executed by other programs.
`/usr/local/`	Local executables and libraries. Also used as the default destination for the FreeBSD ports framework. Within `/usr/local`, the general layout sketched out by hier(7) for `/usr` should be used. Exceptions are the man directory, which is directly under `/usr/local` rather than under `/usr/local/share`, and the ports documentation is in `share/doc/port`.
`/usr/ports/`	The FreeBSD Ports Collection (optional).
`/usr/sbin/`	System daemons and system utilities executed by users.
`/usr/share/`	Architecture-independent files.
`/usr/src/`	BSD and/or local source files.

`/var/`	Multi-purpose log, temporary, transient, and spool files.
`/var/log/`	Miscellaneous system log files.
`/var/tmp/`	Temporary files which are usually preserved across a system reboot.

3.6. Disk Organization

The smallest unit of organization that FreeBSD uses to find files is the filename. Filenames are case-sensitive, which means that `readme.txt` and `README.TXT` are two separate files. FreeBSD does not use the extension of a file to determine whether the file is a program, document, or some other form of data.

Files are stored in directories. A directory may contain no files, or it may contain many hundreds of files. A directory can also contain other directories, allowing a hierarchy of directories within one another in order to organize data.

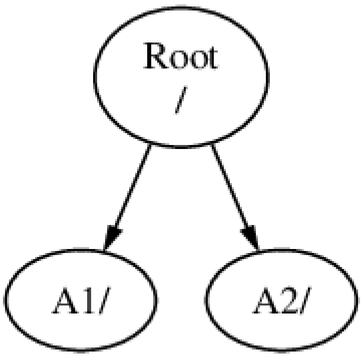
Files and directories are referenced by giving the file or directory name, followed by a forward slash, `/`, followed by any other directory names that are necessary. For example, if the directory `foo` contains a directory `bar` which contains the file `readme.txt`, the full name, or *path*, to the file is `foo/bar/readme.txt`. Note that this is different from Windows® which uses `\` to separate file and directory names. FreeBSD does not use drive letters, or other drive names in the path. For example, one would not type `c:\foo\bar\readme.txt` on FreeBSD.

3.6.1. File systems

Directories and files are stored in a file system. Each file system contains exactly one directory at the very top level, called the *root directory* for that file system. This root directory can contain other directories. One file system is designated the *root file system* or `/`. Every other file system is *mounted* under the root file system. No matter how many disks are on the FreeBSD system, every directory appears to be part of the same disk.

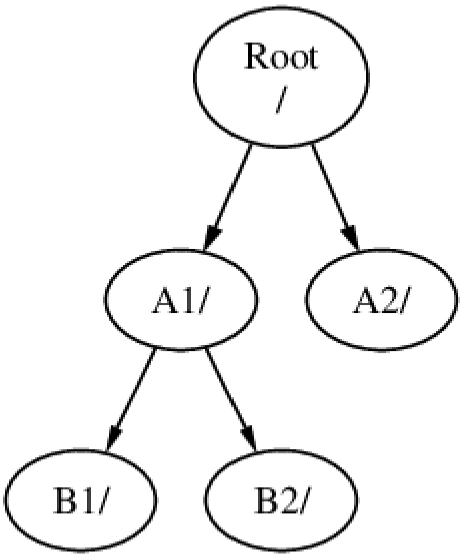
Consider three file systems, called `A`, `B`, and `C`. Each file system has one root directory, which contains two other directories, called `A1`, `A2` (and likewise `B1`, `B2` and `C1`, `C2`).

Call `A` the root file system. If ls(1) is used to view the contents of this directory, it will show two subdirectories, `A1` and `A2`. The directory tree looks like this:



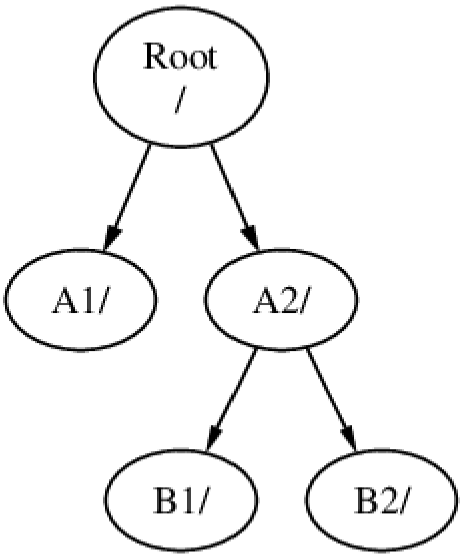
A file system must be mounted on to a directory in another file system. When mounting file system

`B` on to the directory `A1`, the root directory of `B` replaces `A1`, and the directories in `B` appear accordingly:



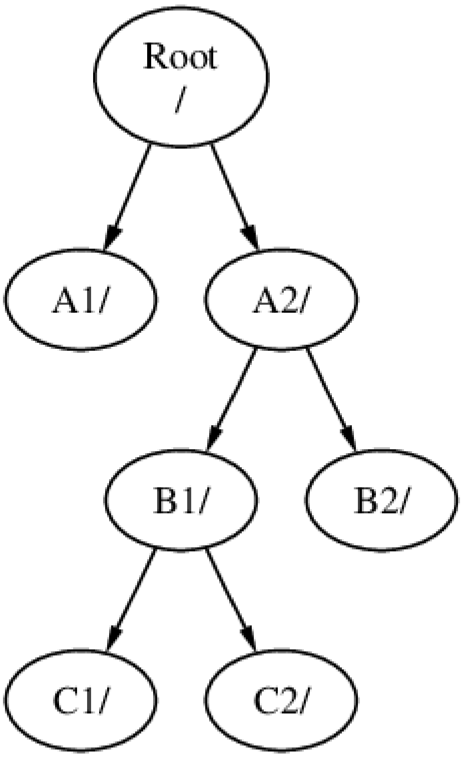
Any files that are in the `B1` or `B2` directories can be reached with the path `/A1/B1` or `/A1/B2` as necessary. Any files that were in `/A1` have been temporarily hidden. They will reappear if `B` is *unmounted* from `A`.

If `B` had been mounted on `A2` then the diagram would look like this:

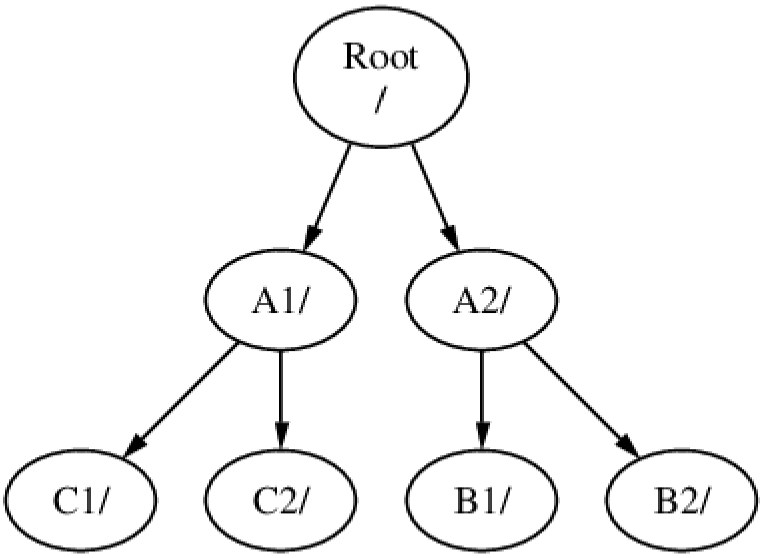


and the paths would be `/A2/B1` and `/A2/B2` respectively.

File systems can be mounted on top of one another. Continuing the last example, the `C` file system could be mounted on top of the `B1` directory in the `B` file system, leading to this arrangement:



Or `C` could be mounted directly on to the `A` file system, under the `A1` directory:



It is entirely possible to have one large root file system, and not need to create any others. There are some drawbacks to this approach, and one advantage.

Benefits of Multiple File Systems

- Different file systems can have different *mount options*. For example, the root file system can be mounted read-only, making it impossible for users to inadvertently delete or edit a critical file. Separating user-writable file systems, such as `/home`, from other file systems allows them to be mounted *nosuid*. This option prevents the *suid*/*guid* bits on executables stored on the file system from taking effect, possibly improving security.
- FreeBSD automatically optimizes the layout of files on a file system, depending on how the file system is being used. So a file system that contains many small files that are written frequently

will have a different optimization to one that contains fewer, larger files. By having one big file system this optimization breaks down.

- FreeBSD’s file systems are robust if power is lost. However, a power loss at a critical point could still damage the structure of the file system. By splitting data over multiple file systems it is more likely that the system will still come up, making it easier to restore from backup as necessary.

Benefit of a Single File System

- File systems are a fixed size. When creating a file system during the FreeBSD installation and giving it a specific size, it may be that one wants to make the partition bigger. This is not easily accomplished without backing up, recreating the file system with the new size, and then restoring the backed up data.

> FreeBSD features the growfs(8) command, which makes it possible to increase the size of file system on the fly, removing this limitation. A file system can only be expanded into free space in the partition in which it resides. If there is space after the partition, the partition can be expanded with gpart(8). If the partition is the last one on a virtual disk, and the disk is expanded, the partition can then be expanded.

3.6.2. Disk partitions

File systems are contained in *partitions*. Disks are divided into partitions using one of several partitioning schemes; see Manual Partitioning. The newer scheme is GPT; older BIOS-based computers use MBR. GPT supports division of a disk into partitions with a size, offset, and type. It supports a large number of partitions and partition types, and is recommended whenever its use is possible. GPT partitions use the disk name with a suffix, where the suffix is `p1` for the first partition, `p2` for the second, and so on. MBR, however, supports only a small number of partitions. The MBR partitions are known in FreeBSD as `slices`. Slices may be used for different operating systems. FreeBSD slices are subdivided into partitions using BSD labels (see bsdlabel(8)).

Slice numbers follow the device name, prefixed with an `s`, starting at 1. So "da0*s1*" is the first slice on the first SCSI drive. There can only be four physical slices on a disk, but there can be logical slices inside physical slices of the appropriate type. These extended slices are numbered starting at 5, so "ada0*s5*" is the first extended slice on the first SATA disk. These devices are used by file systems that expect to occupy a slice.

Each GPT or BSD partition can contain only one file system, which means that file systems are often described by either their typical mount point in the file system hierarchy, or the name of the partition they are contained in.

FreeBSD also uses disk space for *swap space* to provide *virtual memory*. This allows the computer to behave as though it has much more memory than it actually does. When FreeBSD runs out of memory, it moves some of the data that is not currently being used to the swap space, and moves it back in (moving something else out) when it needs it. This is called *paging*.

Some BSD partitions have certain conventions associated with them.

Partition	Convention
`a`	Normally contains the root file system.
`b`	Normally contains swap space.
`c`	Normally the same size as the enclosing slice. This allows utilities that need to work on the entire slice, such as a bad block scanner, to work on the `c` partition. A file system would not normally be created on this partition.
`d`	Partition `d` used to have a special meaning associated with it, although that is now gone and `d` may work as any normal partition.

Slices and "dangerously dedicated" physical drives contain BSD partitions, which are represented as letters from `a` to `h`. This letter is appended to the device name, so "da0*a*" is the `a` partition on the first `da` drive, which is "dangerously dedicated". "ada1s3*e*" is the fifth partition in the third slice of the second SATA disk drive.

Finally, each disk on the system is identified. A disk name starts with a code that indicates the type of disk, and then a number, indicating which disk it is. Unlike partitions and slices, disk numbering starts at 0. Common codes are listed in Disk Device Names.

When referring to a partition in a slice, include the disk name, `s`, the slice number, and then the partition letter. Examples are shown in Sample Disk. GPT partitions include the disk name, `p`, and then the partition number.

Conceptual Model of a Disk shows a conceptual model of a disk layout using MBR slices.

When installing FreeBSD, configure the disk slices if using MBR, and create partitions within the slice to be used for FreeBSD. If using GPT, configure partitions for each file system. In either case, create a file system or swap space in each partition, and decide where each file system will be mounted. See gpart(8) for information on manipulating partitions.

Table 4. Disk Device Names

Drive Type	Drive Device Name
SATA and IDE hard drives	`ada`
SCSI hard drives and USB storage devices	`da`
NVMe storage	`nvd` or `nda`
SATA and IDE CD-ROM drives	`cd`
SCSI CD-ROM drives	`cd`
Floppy drives	`fd`
SCSI tape drives	`sa`
RAID drives	Examples include `aacd` for Adaptec® AdvancedRAID, `mlxd` and `mlyd` for Mylex®, `amrd` for AMI MegaRAID®, `idad` for Compaq Smart RAID, `twed` for 3ware® RAID.

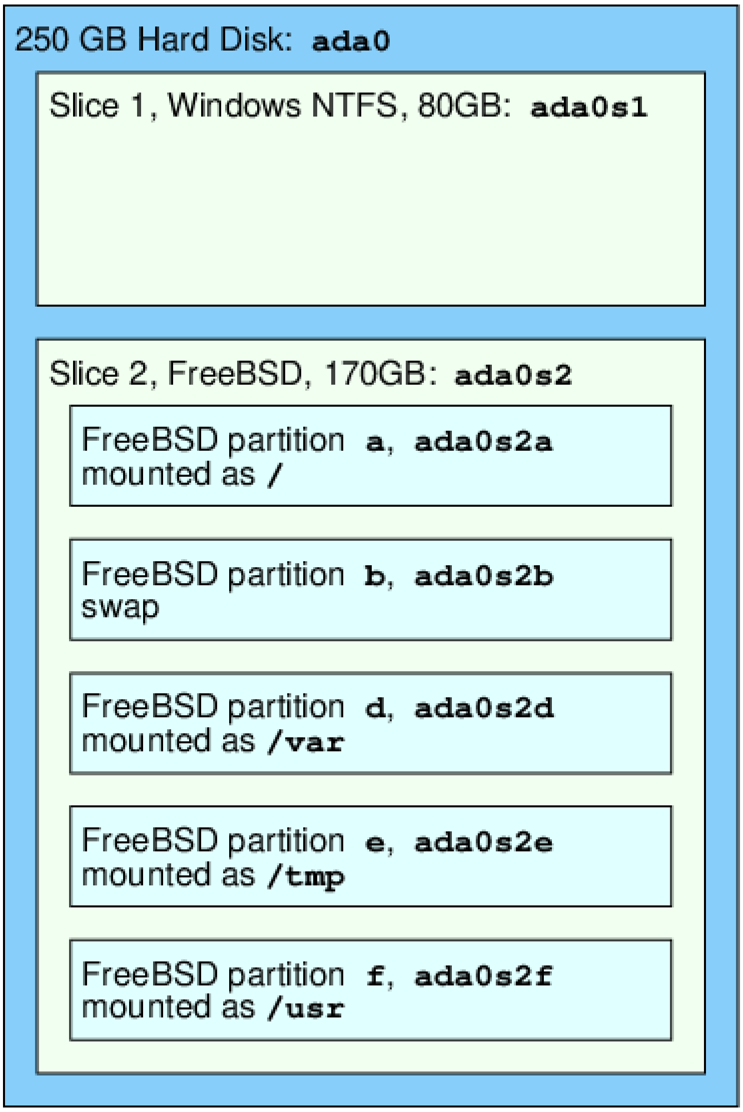
Table 5. Sample Disk, Slice, and Partition Names

Name	Meaning
`ada0s1a`	The first partition (`a`) on the first slice (`s1`) on the first SATA disk (`ada0`).
`da1s2e`	The fifth partition (`e`) on the second slice (`s2`) on the second SCSI disk (`da1`).

Example 13. Conceptual Model of a Disk

This diagram shows FreeBSD's view of the first SATA disk attached to the system. Assume that the disk is 250 GB in size, and contains an 80 GB slice and a 170 GB slice (MS-DOS® partitions). The first slice contains a Windows® NTFS file system, `C:`, and the second slice contains a FreeBSD installation. This example FreeBSD installation has four data partitions and a swap partition.

The four partitions each hold a file system. Partition `a` is used for the root file system, `d` for `/var/`, `e` for `/tmp/`, and `f` for `/usr/`. Partition letter `c` refers to the entire slice, and so is not used for ordinary partitions.


250 GB Hard Disk: ada0
Slice 1, Windows NTFS, 80GB: ada0s1
Slice 2, FreeBSD, 170GB: ada0s2
FreeBSD partition a, ada0s2a mounted as /
FreeBSD partition b, ada0s2b swap
FreeBSD partition d, ada0s2d mounted as /var
FreeBSD partition e, ada0s2e mounted as /tmp
FreeBSD partition f, ada0s2f mounted as /usr

3.7. Mounting and Unmounting File Systems

The file system is best visualized as a tree, rooted, as it were, at `/`. `/dev`, `/usr`, and the other directories in the root directory are branches, which may have their own branches, such as `/usr/local`, and so on.

There are various reasons to house some of these directories on separate file systems. `/var` contains the directories `log/`, `spool/`, and various types of temporary files, and as such, may get filled up. Filling up the root file system is not a good idea, so splitting `/var` from `/` is often favorable.

Another common reason to contain certain directory trees on other file systems is if they are to be housed on separate physical disks, or are separate virtual disks, such as Network File System mounts, described in "Network File System (NFS)", or CDROM drives.

3.7.1. The fstab File

During the boot process (The FreeBSD Booting Process), file systems listed in `/etc/fstab` are automatically mounted except for the entries containing `noauto`. This file contains entries in the following format:

```
device          /mount-point fstype     options      dumpfreq     passno
```

`device`

An existing device name as explained in Disk Device Names.

`mount-point`

An existing directory on which to mount the file system.

`fstype`

The file system type to pass to mount(8). The default FreeBSD file system is `ufs`.

`options`

Either `rw` for read-write file systems, or `ro` for read-only file systems, followed by any other options that may be needed. A common option is `noauto` for file systems not normally mounted during the boot sequence. Other options are listed in mount(8).

`dumpfreq`

Used by dump(8) to determine which file systems require dumping. If the field is missing, a value of zero is assumed.

`passno`

Determines the order in which UFS file systems should be checked by fsck(8) after a reboot. File systems that should be skipped should have their `passno` set to zero. The root file system needs to be checked before everything else and should have its `passno` set to one. The other file systems should be set to values greater than one. If more than one file system has the same `passno`, fsck(8) will attempt to check file systems in parallel if possible.

Refer to fstab(5) for more information on the format of `/etc/fstab` and its options.

3.7.2. Using mount(8)

File systems are mounted using mount(8). The most basic syntax is as follows:

```
# mount device mountpoint
```

A file system listed in `/etc/fstab` can also be mounted by providing just the mountpoint.

This command provides many options which are described in mount(8). The most commonly used options include:

Mount Options

`-a`

Mount all the file systems listed in `/etc/fstab`, except those marked as "noauto", excluded by the `-t` flag, or those that are already mounted.

`-d`

Do everything except for the actual mount system call. This option is useful in conjunction with the `-v` flag to determine what mount(8) is actually trying to do.

`-f`

Force the mount of an unclean file system (dangerous), or the revocation of write access when downgrading a file system’s mount status from read-write to read-only.

`-r`

Mount the file system read-only. This is identical to using `-o ro`.

`-t` **`fstype`**

Mount the specified file system type or mount only file systems of the given type, if `-a` is included. "ufs" is the default file system type.

`-u`

Update mount options on the file system.

`-v`

Be verbose.

`-w`

Mount the file system read-write.

The following options can be passed to `-o` as a comma-separated list:

nosuid

Do not interpret setuid or setgid flags on the file system. This is also a useful security option.

3.7.3. Using umount(8)

To unmount a file system use umount(8). This command takes one parameter which can be a mountpoint, device name, `-a` or `-A`.

All forms take `-f` to force unmounting, and `-v` for verbosity. Be warned that `-f` is not generally a good idea as it might crash the computer or damage data on the file system.

To unmount all mounted file systems, or just the file system types listed after `-t`, use `-a` or `-A`. Note that `-A` does not attempt to unmount the root file system.

3.8. Processes and Daemons

FreeBSD is a multi-tasking operating system. Each program running at any one time is called a *process*. Every running command starts at least one new process and there are a number of system processes that are run by FreeBSD.

Each process is uniquely identified by a number called a *process ID* (PID). Similar to files, each process has one owner and group, and the owner and group permissions are used to determine which files and devices the process can open. Most processes also have a parent process that started them. For example, the shell is a process, and any command started in the shell is a process which has the shell as its parent process. The exception is a special process called init(8) which is always the first process to start at boot time and which always has a PID of `1`.

Some programs are not designed to be run with continuous user input and disconnect from the terminal at the first opportunity. For example, a web server responds to web requests, rather than user input. Mail servers are another example of this type of application. These types of programs are known as *daemons*. The term daemon comes from Greek mythology and represents an entity that is neither good nor evil, and which invisibly performs useful tasks. This is why the BSD mascot is the cheerful-looking daemon with sneakers and a pitchfork.

There is a convention to name programs that normally run as daemons with a trailing "d". For example, BIND is the Berkeley Internet Name Domain, but the actual program that executes is `named`. The Apache web server program is `httpd` and the line printer spooling daemon is `lpd`. This is only a naming convention. For example, the main mail daemon for the Sendmail application is `sendmail`, and not `maild`.

3.8.1. Viewing Processes

To see the processes running on the system, use ps(1) or top(1). To display a static list of the currently running processes, their PIDs, how much memory they are using, and the command they were started with, use ps(1). To display all the running processes and update the display every few seconds in order to interactively see what the computer is doing, use top(1).

By default, ps(1) only shows the commands that are running and owned by the user. For example:

```
% ps
```

The output should be similar to the following:

```
 PID TT  STAT    TIME COMMAND
8203  0  Ss   0:00.59 /bin/csh
8895  0  R+   0:00.00 ps
```

The output from ps(1) is organized into a number of columns. The `PID` column displays the process ID. PIDs are assigned starting at 1, go up to 99999, then wrap around back to the beginning. However, a PID is not reassigned if it is already in use. The `TT` column shows the tty the program is running on and `STAT` shows the program's state. `TIME` is the amount of time the program has been running on the CPU. This is usually not the elapsed time since the program was started, as most programs spend a lot of time waiting for things to happen before they need to spend time on the CPU. Finally, `COMMAND` is the command that was used to start the program.

A number of different options are available to change the information that is displayed. One of the most useful sets is `auxww`, where `a` displays information about all the running processes of all users, `u` displays the username and memory usage of the process' owner, `x` displays information about daemon processes, and `ww` causes ps(1) to display the full command line for each process, rather than truncating it once it gets too long to fit on the screen.

The output from top(1) is similar:

```
% top
```

The output should be similar to the following:

```
last pid:  9609;  load averages:  0.56,  0.45,  0.36              up 0+00:20:03
10:21:46
107 processes: 2 running, 104 sleeping, 1 zombie
CPU:  6.2% user,  0.1% nice,  8.2% system,  0.4% interrupt, 85.1% idle
Mem: 541M Active, 450M Inact, 1333M Wired, 4064K Cache, 1498M Free
ARC: 992M Total, 377M MFU, 589M MRU, 250K Anon, 5280K Header, 21M Other
Swap: 2048M Total, 2048M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME   WCPU COMMAND
  557 root          1 -21  r31   136M 42296K select  0   2:20  9.96% Xorg
 8198 dru           2  52    0   449M 82736K select  3   0:08  5.96% kdeinit4
 8311 dru          27  30    0  1150M   187M uwait   1   1:37  0.98% firefox
  431 root          1  20    0 14268K  1728K select  0   0:06  0.98% moused
 9551 dru           1  21    0 16600K  2660K CPU3    3   0:01  0.98% top
 2357 dru           4  37    0   718M   141M select  0   0:21  0.00% kdeinit4
 8705 dru           4  35    0   480M    98M select  2   0:20  0.00% kdeinit4
 8076 dru           6  20    0   552M   113M uwait   0   0:12  0.00% soffice.bin
 2623 root          1  30   10 12088K  1636K select  3   0:09  0.00% powerd
 2338 dru           1  20    0   440M 84532K select  1   0:06  0.00% kwin
 1427 dru           5  22    0   605M 86412K select  1   0:05  0.00% kdeinit4
```

The output is split into two sections. The header (the first five or six lines) shows the PID of the last process to run, the system load averages (which are a measure of how busy the system is), the system uptime (time since the last reboot) and the current time. The other figures in the header relate to how many processes are running, how much memory and swap space has been used, and how much time the system is spending in different CPU states. If the ZFS file system module has been loaded, an `ARC` line indicates how much data was read from the memory cache instead of from disk.

Below the header is a series of columns containing similar information to the output from ps(1), such as the PID, username, amount of CPU time, and the command that started the process. By default, top(1) also displays the amount of memory space taken by the process. This is split into two columns: one for total size and one for resident size. Total size is how much memory the application has needed and the resident size is how much it is actually using now.

top(1) automatically updates the display every two seconds. A different interval can be specified with `-s`.

3.8.2. Killing Processes

One way to communicate with any running process or daemon is to send a *signal* using kill(1). There are a number of different signals; some have a specific meaning while others are described in the application's documentation. A user can only send a signal to a process they own and sending a signal to someone else's process will result in a permission denied error. The exception is the `root` user, who can send signals to anyone's processes.

The operating system can also send a signal to a process. If an application is badly written and tries to access memory that it is not supposed to, FreeBSD will send the process the "Segmentation Violation" signal (`SIGSEGV`). If an application has been written to use the alarm(3) system call to be alerted after a period of time has elapsed, it will be sent the "Alarm" signal (`SIGALRM`).

Two signals can be used to stop a process: `SIGTERM` and `SIGKILL`. `SIGTERM` is the polite way to kill a process as the process can read the signal, close any log files it may have open, and attempt to finish what it is doing before shutting down. In some cases, a process may ignore `SIGTERM` if it is in the middle of some task that cannot be interrupted.

`SIGKILL` cannot be ignored by a process. Sending a `SIGKILL` to a process will usually stop that process there and then. [1].

Other commonly used signals are `SIGHUP`, `SIGUSR1`, and `SIGUSR2`. Since these are general purpose signals, different applications will respond differently.

For example, after changing a web server's configuration file, the web server needs to be told to re-read its configuration. Restarting `httpd` would result in a brief outage period on the web server. Instead, send the daemon the `SIGHUP` signal. Be aware that different daemons will have different behavior, so refer to the documentation for the daemon to determine if `SIGHUP` will achieve the desired results.

Killing a random process on the system is a bad idea. In particular, init(8), PID 1, is special. Running `/bin/kill -s KILL 1` is a quick, and unrecommended, way to

> shutdown the system. *Always* double check the arguments to kill(1) *before* pressing `Return`.

3.9. Shells

A *shell* provides a command line interface for interacting with the operating system. A shell receives commands from the input channel and executes them. Many shells provide built in functions to help with everyday tasks such as file management, file globbing, command line editing, command macros, and environment variables. FreeBSD comes with several shells, including an extended POSIX® shell (sh(1)) and the extended C shell (tcsh(1)). Other shells are available from the FreeBSD Ports Collection, such as `zsh` and `bash`.

The shell that is used is really a matter of taste. A C programmer might feel more comfortable with a C-like shell such as tcsh(1). A Linux® user might prefer `bash`. Each shell has unique properties that may or may not work with a user's preferred working environment, which is why there is a choice of which shell to use.

One common shell feature is filename completion. After a user types the first few letters of a command or filename and presses `Tab`, the shell completes the rest of the command or filename. Consider two files called `foobar` and `football`. To delete `foobar`, the user might type `rm foo` and press `Tab` to complete the filename.

But the shell only shows `rm foo`. It was unable to complete the filename because both `foobar` and `football` start with `foo`. Some shells sound a beep or show all the choices if more than one name matches. The user must then type more characters to identify the desired filename. Typing a `t` and pressing `Tab` again is enough to let the shell determine which filename is desired and fill in the rest.

Another feature of the shell is the use of environment variables. Environment variables are a variable/key pair stored in the shell's environment. This environment can be read by any program invoked by the shell, and thus contains a lot of program configuration. Common Environment Variables provides a list of common environment variables and their meanings. Note that the names of environment variables are always in uppercase.

Table 6. Common Environment Variables

Variable	Description
`USER`	Current logged in user's name.
`PATH`	Colon-separated list of directories to search for binaries.
`DISPLAY`	Network name of the Xorg display to connect to, if available.
`SHELL`	The current shell.
`TERM`	The name of the user's type of terminal. Used to determine the capabilities of the terminal.
`TERMCAP`	Database entry of the terminal escape codes to perform various terminal functions.
`OSTYPE`	Type of operating system.
`MACHTYPE`	The system's CPU architecture.

`EDITOR`	The user's preferred text editor.
`PAGER`	The user's preferred utility for viewing text one page at a time.
`MANPATH`	Colon-separated list of directories to search for manual pages.

How to set an environment variable differs between shells. In tcsh(1) and csh(1), use `setenv` to set environment variables. In sh(1) and `bash`, use `export` to set the current environment variables. This example sets the default `EDITOR` to `/usr/local/bin/emacs` for the tcsh(1) shell:

```
% setenv EDITOR /usr/local/bin/emacs
```

The equivalent command for `bash` would be:

```
% export EDITOR="/usr/local/bin/emacs"
```

To expand an environment variable in order to see its current setting, type a `$` character in front of its name on the command line. For example, `echo $TERM` displays the current `$TERM` setting.

Shells treat special characters, known as meta-characters, as special representations of data. The most common meta-character is `*`, which represents any number of characters in a filename. Meta-characters can be used to perform filename globbing. For example, `echo *` is equivalent to `ls` because the shell takes all the files that match `*` and `echo` lists them on the command line.

To prevent the shell from interpreting a special character, escape it from the shell by starting it with a backslash (`\`). For example, `echo $TERM` prints the terminal setting whereas `echo \$TERM` literally prints the string `$TERM`.

3.9.1. Changing the Shell

The easiest way to permanently change the default shell is to use `chsh`. Running this command will open the editor that is configured in the `EDITOR` environment variable, which by default is set to vi(1). Change the `Shell:` line to the full path of the new shell.

Alternately, use `chsh -s` which will set the specified shell without opening an editor. For example, to change the shell to `bash`:

```
% chsh -s /usr/local/bin/bash
```

Enter the password at the prompt and press `Return` to change the shell. Log off and log in again to start using the new shell.

The new shell *must* be present in `/etc/shells`. If the shell was installed from the FreeBSD Ports Collection as described in Installing Applications: Packages and Ports, it should be automatically added to this file. If it is missing, add it using this command, replacing the path with the path of the shell:

```
# echo /usr/local/bin/bash >> /etc/shells
```

Then, rerun chsh(1).

3.9.2. Advanced Shell Techniques

The UNIX® shell is not just a command interpreter, it acts as a powerful tool which allows users to execute commands, redirect their output, redirect their input and chain commands together to improve the final command output. When this functionality is mixed with built in commands, the user is provided with an environment that can maximize efficiency.

Shell redirection is the action of sending the output or the input of a command into another command or into a file. To capture the output of the ls(1) command, for example, into a file, redirect the output:

```
% ls > directory_listing.txt
```

The directory contents will now be listed in `directory_listing.txt`. Some commands can be used to read input, such as sort(1). To sort this listing, redirect the input:

```
% sort < directory_listing.txt
```

The input will be sorted and placed on the screen. To redirect that input into another file, one could redirect the output of sort(1) by mixing the direction:

```
% sort < directory_listing.txt > sorted.txt
```

In all of the previous examples, the commands are performing redirection using file descriptors. Every UNIX® system has file descriptors, which include standard input (stdin), standard output (stdout), and standard error (stderr). Each one has a purpose, where input could be a keyboard or a mouse, something that provides input. Output could be a screen or paper in a printer. And error would be anything that is used for diagnostic or error messages. All three are considered I/O based file descriptors and sometimes considered streams.

Through the use of these descriptors, the shell allows output and input to be passed around through various commands and redirected to or from a file. Another method of redirection is the pipe operator.

The UNIX® pipe operator, "|" allows the output of one command to be directly passed or directed to another program. Basically, a pipe allows the standard output of a command to be passed as standard input to another command, for example:

```
% cat directory_listing.txt | sort | less
```

In that example, the contents of `directory_listing.txt` will be sorted and the output passed to less(1). This allows the user to scroll through the output at their own pace and prevent it from scrolling off the screen.

3.10. Text Editors

Most FreeBSD configuration is done by editing text files, so it is a good idea to become familiar with a text editor. FreeBSD comes with a few as part of the base system, and many more are available in the Ports Collection.

A simple editor to learn is ee(1), which stands for easy editor. To start this editor, type `ee filename` where *filename* is the name of the file to be edited. Once inside the editor, all of the commands for manipulating the editor's functions are listed at the top of the display. The caret (`^`) represents `Ctrl`, so `^e` expands to `Ctrl` + `e`. To leave ee(1), press `Esc`, then choose the "leave editor" option from the main menu. The editor will prompt to save any changes if the file has been modified.

FreeBSD also comes with more powerful text editors, such as vi(1), as part of the base system. Other editors, like editors/emacs and editors/vim, are part of the FreeBSD Ports Collection. These editors offer more functionality at the expense of being more complicated to learn. Learning a more powerful editor such as vim or Emacs can save more time in the long run.

Many applications which modify files or require typed input will automatically open a text editor. To change the default editor, set the `EDITOR` environment variable as described in Shells.

3.11. Devices and Device Nodes

A device is a term used mostly for hardware-related activities in a system, including disks, printers, graphics cards, and keyboards. When FreeBSD boots, the majority of the boot messages refer to devices being detected. A copy of the boot messages is saved to `/var/run/dmesg.boot`.

Each device has a device name and number. For example, `ada0` is the first SATA hard drive, while `kbd0` represents the keyboard.

Most devices in FreeBSD must be accessed through special files called device nodes, which are located in `/dev`.

3.12. Manual Pages

The most comprehensive documentation on FreeBSD is in the form of manual pages. Nearly every program on the system comes with a short reference manual explaining the basic operation and available arguments. These manuals can be viewed using `man`:

```
% man command
```

where *command* is the name of the command to learn about. For example, to learn more about ls(1), type:

```
% man ls
```

Manual pages are divided into sections which represent the type of topic. In FreeBSD, the following sections are available:

1. User commands.
2. System calls and error numbers.
3. Functions in the C libraries.
4. Device drivers.
5. File formats.
6. Games and other diversions.
7. Miscellaneous information.
8. System maintenance and operation commands.
9. System kernel interfaces.

In some cases, the same topic may appear in more than one section of the online manual. For example, there is a `chmod` user command and a `chmod()` system call. To tell man(1) which section to display, specify the section number:

```
% man 1 chmod
```

This will display the manual page for the user command chmod(1). References to a particular section of the online manual are traditionally placed in parenthesis in written documentation, so chmod(1) refers to the user command and chmod(2) refers to the system call.

If the name of the manual page is unknown, use `man -k` to search for keywords in the manual page descriptions:

```
% man -k mail
```

This command displays a list of commands that have the keyword "mail" in their descriptions. This is equivalent to using apropos(1).

To read the descriptions for all of the commands in `/usr/sbin`, type:

```
% cd /usr/sbin
% man -f * | more
```

or

```
% cd /usr/sbin
% whatis * |more
```

3.12.1. GNU Info Files

FreeBSD includes several applications and utilities produced by the Free Software Foundation (FSF). In addition to manual pages, these programs may include hypertext documents called `info` files. These can be viewed using info(1) or, if editors/emacs is installed, the info mode of emacs.

To use info(1), type:

```
% info
```

For a brief introduction, type `h`. For a quick command reference, type `?`.

---

[1] There are a few tasks that cannot be interrupted. For example, if the process is trying to read from a file that is on another computer on the network, and the other computer is unavailable, the process is said to be uninterruptible. Eventually the process will time out, typically after two minutes. As soon as this time out occurs the process will be killed.

Chapter 4. Installing Applications: Packages and Ports

4.1. Synopsis

FreeBSD is bundled with a rich collection of system tools as part of the base system. In addition, FreeBSD provides two complementary technologies for installing third-party software: the FreeBSD Ports Collection, for installing from source, and packages, for installing from pre-built binaries. Either method may be used to install software from local media or from the network.

Read this chapter to learn:

- The difference between binary packages and ports.
- How to find third-party software that has been ported to FreeBSD.
- How to manage binary packages using pkg.
- How to build third-party software from source using the Ports Collection.
- How to find the files installed with the application for post-installation configuration.
- What to do if a software installation fails.

4.2. Overview of Software Installation

A FreeBSD *port* is a collection of files designed to automate the process of compiling an application from source code. The files that comprise a port contain all the necessary information to automatically download, extract, patch, compile, and install the application.

If the software has not already been adapted and tested on FreeBSD, the source code might need editing in order for it to install and run properly.

However, over 36000 third-party applications have already been ported to FreeBSD. When feasible, these applications are made available for download as pre-compiled *packages*.

Packages can be manipulated with the FreeBSD package management commands.

Both packages and ports understand dependencies. If a package or port is used to install an application and a dependent library is not already installed, the library will automatically be installed first.

A FreeBSD package contains pre-compiled copies of all the commands for an application, as well as any configuration files and documentation. A package can be manipulated with the pkg(8) commands, such as `pkg install`.

While the two technologies are similar, packages and ports each have their own strengths. Select the technology that meets the needs for installing a particular application.

Package Benefits

- A compressed package tarball is typically smaller than the compressed tarball containing the source code for the application.
- Packages do not require compilation time. For large applications, such as Firefox, KDE Plasma, or GNOME, this can be important on a slow system.
- Packages do not require any understanding of the process involved in compiling software on FreeBSD.

Port Benefits

- Packages are normally compiled with conservative options because they have to run on the maximum number of systems. By compiling from the port, one can change the compilation options.
- Some applications have compile-time options relating to which features are installed. For example, NGINX® can be configured with a wide variety of different built-in options.

  In some cases, multiple packages will exist for the same application with different settings. For example, NGINX® is available as a `nginx` package and a `nginx-lite` package, the former has many more options enabled, but this in turn requires many things to be installed as dependencies for it to work, thus increasing space consumption and attack surface.

  The transitive dependencies can grow quite large, for example the full `nginx` package will pull in several X libraries which can be quite surprising, so building from ports allows to choose only the options needed without a "kitchen sink" approach. In some cases, multiple packages will exist for the same application to specify certain settings.
- The licensing conditions of some software forbid binary distribution. Such software must be distributed as source code which must be compiled by the end-user.
- Some people do not trust binary distributions or prefer to read through source code in order to look for potential problems.
- Source code is needed in order to apply custom patches.

To keep track of updated ports, subscribe to the FreeBSD ports mailing list and the FreeBSD ports bugs mailing list.

> Before installing an application, check https://vuxml.freebsd.org/ for related security issues.
>
> To audit installed packages against known vulnerabilities, run `pkg audit -F`.

The remainder of this chapter explains how to use packages and ports to install and manage third-party software on FreeBSD.

4.3. Finding Software

FreeBSD's list of available applications is growing all the time. There are a number of ways to find software to install:

- The FreeBSD web site maintains an up-to-date searchable list of all the available applications, at

Ports Portal. The ports can be searched by application name or by software category.

- Dan Langille maintains FreshPorts which provides a comprehensive search utility and also tracks changes to the applications in the Ports Collection. Registered users can create a customized watch list in order to receive an automated email when their watched ports are updated.
- If finding a particular application becomes challenging, try searching a site like SourceForge or GitHub then check back at the Ports Portal to see if the application has been ported.
- Search the binary package repository for an application using the pkg(8) command

4.4. Using pkg for Binary Package Management

pkg(8) provides an interface for manipulating packages: registering, adding, removing and upgrading packages.

For sites wishing to only use prebuilt binary packages from the FreeBSD mirrors, managing packages with pkg(8) can be sufficient.

However, for those sites building from source a separate port management tool will be needed.

Since pkg(8) only works with binary packages, it is not a replacement for such tools. Those tools can be used to install software from both binary packages and the Ports Collection, while pkg(8) installs only binary packages.

4.4.1. Getting Started with pkg

All supported versions of FreeBSD now contain `/usr/sbin/pkg` a.k.a pkg(7). This is a small placeholder that has just the minimum functionality required to install the real pkg(8).

An Internet working connection is required for the bootstrap process to succeed.

Run pkg(8) command line:

```
# pkg
```

The output should be similar to the following:

```
The package management tool is not yet installed on your system.
Do you want to fetch and install it now? [y/N]
```

pkg(7) will intercept the command, and when confirmed, downloads the pkg(8) tarball, installs pkg(8) from it, bootstraps the local package database and then proceeds to run the command originally requested.

More recent versions of pkg(7) understand `pkg -N` as a test to see if pkg(8) is installed without triggering the installation, and conversely, pkg bootstrap[-f] to install pkg(8) (or force it to be reinstalled) without performing any other actions.

Usage information for pkg is available in the pkg(8) manual page or by running `pkg` without additional arguments. Additional pkg configuration options are described in pkg.conf(5).

Each pkg command argument is documented in a command-specific manual page.

To read the manual page for `pkg install`, for example, run this command:

```
# pkg help install
```

The rest of this section demonstrates common binary package management tasks which can be performed using pkg(8). Each demonstrated command provides many switches to customize its use. Refer to a command's help or man page for details and more examples.

4.4.2. Quarterly and Latest Ports Branches

The `Quarterly` branch provides users with a more predictable and stable experience for port and package installation and upgrades. This is done essentially by only allowing non-feature updates. Quarterly branches aim to receive security fixes (that may be version updates, or backports of commits), bug fixes and ports compliance or framework changes. The Quarterly branch is cut from HEAD at the beginning of every (yearly) quarter in January, April, July, and October. Branches are named according to the year (YYYY) and quarter (Q1-4) they are created in. For example, the quarterly branch created in January 2023, is named 2023Q1. And the `Latest` branch provides the latest versions of the packages to the users.

To switch pkg(8) from Quarterly to Latest run the following commands:

```
# mkdir -p /usr/local/etc/pkg/repos
# echo 'FreeBSD-ports: { url: "pkg+https://pkg.FreeBSD.org/${ABI}/latest" }' >
/usr/local/etc/pkg/repos/FreeBSD.conf
```

Then run this command to update the local package repositories catalogues for the Latest branch:

```
# pkg update -f
```

4.4.3. Kernel modules repositories

The kernel module repositories allow users to install ready to use kernel modules like the ones used for graphic drivers and specific hardware. Starting with FreeBSD 14.3, the FreeBSD Project provides kernel modules built for every supported version. To create such a repository configuration if it does not exist, add the following to /usr/local/etc/pkg/repos/kmods.conf:

```
FreeBSD-kmods: {
        url: "pkg+https://pkg.FreeBSD.org/${ABI}/KMODSFLAVOR",
        mirror_type: "srv",
        signature_type: "fingerprints",
        fingerprints: "/usr/share/keys/pkg",
        enabled: yes
}
```

`KMODSFLAVOR` respects the following pattern: `kmods_PORTBRANCH_MINORRELEASE`.

For instance:

Table 7. Kmodsflavor

FreeBSD Release	ports main	ports quarterly
FreeBSD 14.2-RELEASE	kmods_latest_2	kmods_quarterly_2
FreeBSD 14.3-RELEASE	kmods_latest_3	kmods_quarterly_3
FreeBSD 14.3-STABLE	kmods_latest	kmods_quarterly
FreeBSD 15.0-CURRENT	kmods_latest	

4.4.4. Configure pkg

pkg.conf(5) is the system-wide configuration file used by the pkg(8) tools. The default location of this file is `/usr/local/etc/pkg.conf`.

> FreeBSD does not need to have a `pkg.conf` file. Many installations will work well with no `pkg.conf` at all or with an empty `pkg.conf` (other than comment lines).

Lines in the file beginning with a "#" are comments and are ignored.

The file is in UCL format. For more information on the syntax of libucl(3), please visit the official UCL website.

The following types of options are recognized - boolean, string and list options.

A boolean option is marked as enabled if one of the following values is specified in the configuration file - YES, TRUE and ON.

4.4.5. Searching Packages

To search a package pkg-search(8) can be used:

```
# pkg search nginx
```

The output should be similar to the following:

```
modsecurity3-nginx-1.0.3       Instruction detection and prevention engine / nginx
Wrapper
nginx-1.22.1_2,3               Robust and small WWW server
nginx-devel-1.23.2_4           Robust and small WWW server
nginx-full-1.22.1_1,3          Robust and small WWW server (full package)
nginx-lite-1.22.1,3            Robust and small WWW server (lite package)
```

```
nginx-naxsi-1.22.1,3           Robust and small WWW server (plus NAXSI)
nginx-prometheus-exporter-0.10.0_7 Prometheus exporter for NGINX and NGINX Plus stats
nginx-ultimate-bad-bot-blocker-4.2020.03.2005_1 Nginx bad bot and other things blocker
nginx-vts-exporter-0.10.7_7    Server that scraps NGINX vts stats and export them via
HTTP
p5-Nginx-ReadBody-0.07_1       Nginx embeded perl module to read and evaluate a
request body
p5-Nginx-Simple-0.07_1         Perl 5 module for easy to use interface for Nginx Perl
Module
p5-Test-Nginx-0.30             Testing modules for Nginx C module development
py39-certbot-nginx-2.0.0       NGINX plugin for Certbot
rubygem-passenger-nginx-6.0.15 Modules for running Ruby on Rails and Rack applications
```

4.4.6. Installing and Fetching Packages

To install a binary package pkg-install(8) can be used. This command uses repository data to determine which version of the software to install and if it has any uninstalled dependencies. For example, to install curl:

```
# pkg install curl
```

The output should be similar to the following:

```
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
All repositories are up to date.
The following 9 package(s) will be affected (of 0 checked):

New packages to be INSTALLED:
        ca_root_nss: 3.83
        curl: 7.86.0
        gettext-runtime: 0.21
        indexinfo: 0.3.1
        libidn2: 2.3.3
        libnghttp2: 1.48.0
        libpsl: 0.21.1_4
        libssh2: 1.10.0.3
        libunistring: 1.0

Number of packages to be installed: 9

The process will require 11 MiB more space.
3 MiB to be downloaded

Proceed with this action? [y/N]
```

The new package and any additional packages that were installed as dependencies can be seen in the installed packages list:

```
# pkg info
```

The output should be similar to the following:

```
ca_root_nss-3.83               Root certificate bundle from the Mozilla Project
curl-7.86.0                    Command line tool and library for transferring data
with URLs
gettext-runtime-0.21.1         GNU gettext runtime libraries and programs
indexinfo-0.3.1                Utility to regenerate the GNU info page index
libidn2-2.3.3                  Implementation of IDNA2008 internationalized domain
names
libnghttp2-1.48.0              HTTP/2.0 C Library
libpsl-0.21.1_6                C library to handle the Public Suffix List
libssh2-1.10.0.3               Library implementing the SSH2 protocol
libunistring-1.0               Unicode string library
pkg-1.18.4                     Package manager
```

To fetch a package and install it later or in another place use pkg-fetch(8). For example, to download `nginx-lite`:

```
# pkg fetch -d -o /usr/home/user/packages/ nginx-lite
```

- `-d`: used to fetch all the dependencies
- `-o`: used to specify the download directory

The output should be similar to the following:

```
Updating FreeBSD repository catalogue...
FreeBSD repository is up to date.
All repositories are up to date.
The following packages will be fetched:

New packages to be FETCHED:
        nginx-lite: 1.22.1,3 (342 KiB: 22.20% of the 2 MiB to download)
        pcre: 8.45_3 (1 MiB: 77.80% of the 2 MiB to download)

Number of packages to be fetched: 2

The process will require 2 MiB more space.
2 MiB to be downloaded.

Proceed with fetching packages? [y/N]:
```

To install the downloaded packages pkg-install(8) can be used as follows:

```
# cd /usr/home/user/packages/
```

```
# pkg install nginx-lite-1.22.1,3.pkg
```

4.4.7. Obtaining Information About Installed Packages

Information about the packages installed on a system can be viewed by running pkg-info(8) which, when run without any switches, will list the package version for either all installed packages or the specified package.

For example, to see which version of pkg is installed, run:

```
# pkg info pkg
```

The output should be similar to the following:

```
pkg-1.19.0
Name           : pkg
Version        : 1.19.0
Installed on   : Sat Dec 17 11:05:28 2022 CET
Origin         : ports-mgmt/pkg
Architecture   : FreeBSD:13:amd64
Prefix         : /usr/local
Categories     : ports-mgmt
Licenses       : BSD2CLAUSE
Maintainer     : pkg@FreeBSD.org
WWW            : https://github.com/freebsd/pkg
Comment        : Package manager
Options        :
        DOCS           : on
Shared Libs provided:
        libpkg.so.4
Annotations    :
        FreeBSD_version: 1301000
        repo_type      : binary
        repository     : FreeBSD
Flat size      : 33.2MiB
Description    :
Package management tool

WWW: https://github.com/freebsd/pkg
```

4.4.8. Upgrading Installed Packages

Installed packages can be upgraded to their latest versions using pkg-upgrade(8):

```
# pkg upgrade
```

This command will compare the installed versions with those available in the repository catalogue and upgrade them from the repository.

4.4.9. Auditing Installed Packages

Software vulnerabilities are regularly discovered in third-party applications. To address this, pkg includes a built-in auditing mechanism. To determine if there are any known vulnerabilities for the software installed on the system, use pkg-audit(8):

```
# pkg audit -F
```

The output should be similar to the following:

```
Fetching vuln.xml.xz: 100%  976 KiB 499.5kB/s    00:02
chromium-108.0.5359.98 is vulnerable:
  chromium -- multiple vulnerabilities
  CVE: CVE-2022-4440
  CVE: CVE-2022-4439
  CVE: CVE-2022-4438
  CVE: CVE-2022-4437
  CVE: CVE-2022-4436
  WWW: https://vuxml.FreeBSD.org/freebsd/83eb9374-7b97-11ed-be8f-3065ec8fd3ec.html
```

4.4.10. Removing Packages

Packages that are no longer needed can be removed with pkg-delete(8).

For example:

```
# pkg delete curl
```

The output should be similar to the following:

```
Checking integrity... done (0 conflicting)
Deinstallation has been requested for the following 1 packages (of 0 packages in the
universe):

Installed packages to be REMOVED:
        curl :7.86.0

Number of packages to be removed: 1

The operation will free 4 MiB.
```

```
Proceed with deinstallation packages? [y/N]: y
[1/1] Deinstalling curl-7.86.0...
[1/1] Deleting files for curl-7.86.0: 100%
```

4.4.11. Automatically Removing Unused Packages

Removing a package may leave behind dependencies which are no longer required. Unneeded packages that were installed as dependencies (leaf packages) can be automatically detected and removed using pkg-autoremove(8):

```
# pkg autoremove
```

The output should be similar to the following:

```
Checking integrity... done (0 conflicting)
Deinstallation has been requested for the following 1 packages:

Installed packages to be REMOVED:
        ca_root_nss-3.83

Number of packages to be removed: 1

The operation will free 723 KiB.

Proceed with deinstalling packages? [y/N]:
```

Packages installed as dependencies are called *automatic* packages. Non-automatic packages, i.e the packages that were explicitly installed not as a dependency to another package, can be listed using:

```
# pkg prime-list
```

The output should be similar to the following:

```
nginx
openvpn
sudo
```

`pkg prime-list` is an alias command declared in `/usr/local/etc/pkg.conf`. There are many others that can be used to query the package database of the system. For instance, command `pkg prime-origins` can be used to get the origin port directory of the list mentioned above:

```
# pkg prime-origins
```

The output should be similar to the following:

```
www/nginx
security/openvpn
security/sudo
```

This list can be used to rebuild all packages installed on a system using build tools such as ports-mgmt/poudriere or ports-mgmt/synth.

Marking an installed package as automatic can be done using:

```
# pkg set -A 1 devel/cmake
```

Once a package is a leaf package and is marked as automatic, it gets selected by `pkg autoremove`.

Marking an installed package as *not* automatic can be done using:

```
# pkg set -A 0 devel/cmake
```

4.4.12. Removing Stale Packages

By default, pkg stores binary packages in a cache directory defined by `PKG_CACHEDIR` in pkg.conf(5). Only copies of the latest installed packages are kept. Older versions of pkg kept all previous packages. To remove these outdated binary packages, run:

```
# pkg clean
```

The entire cache may be cleared by running:

```
# pkg clean -a
```

4.4.13. Locking and Unlocking Packages

pkg-lock(8) is used to lock packages against reinstallation, modification or deletion. pkg-unlock(8) unlocks the named packages. Either variant only has an effect on currently installed packages. Consequently it is impossible to block installation of a new package by using this mechanism, unless such an installation implies updating a locked package.

For example, to lock `nginx-lite`:

```
# pkg lock nginx-lite
```

And to unlock `nginx-lite`:

```
# pkg unlock nginx-lite
```

4.4.14. Modifying Package Metadata

Software within the FreeBSD Ports Collection can undergo major version number changes. To address this, pkg has a built-in command to update package origins. This can be useful, for example, if lang/python3 is renamed to lang/python311 so that lang/python3 can now represent version `3.11`.

To change the package origin for the above example, run:

```
# pkg set -o lang/python3:lang/python311
```

As another example, to update lang/ruby31 to lang/ruby32, run:

```
# pkg set -o lang/ruby31:lang/ruby32
```

> When changing package origins, it is important to reinstall packages that are dependent on the package with the modified origin. To force a reinstallation of dependent packages, run:
>
> ```
> # pkg install -Rf lang/ruby32
> ```

4.5. Using the Ports Collection

The Ports Collection is a set of `Makefiles`, patches, and description files. Each set of these files is used to compile and install an individual application on FreeBSD, and is called a *port*.

By default, the Ports Collection itself is stored as a subdirectory of `/usr/ports`.

> Before installing and using the Ports Collection, please be aware that it is generally ill-advised to use the Ports Collection in conjunction with the binary packages provided via pkg to install software. pkg, by default, tracks quarterly branch-releases of the ports tree and not HEAD. Dependencies could be different for a port in HEAD compared to its counterpart in a quarterly branch release and this could result in conflicts between dependencies installed by pkg and those from the Ports Collection. If the Ports Collection and pkg must be used in conjunction, then be sure that the Ports Collection and pkg are on the same branch release of the ports tree.

The Ports Collection contains directories for software categories. Inside each category are subdirectories for individual applications. Each application subdirectory contains a set of files that tells FreeBSD how to compile and install that program, called a *ports skeleton*. Each port skeleton includes these files and directories:

- **Makefile**: contains statements that specify how the application should be compiled and where its components should be installed.
- **distinfo**: contains the names and checksums of the files that must be downloaded to build the port.
- **files/**: this directory contains any patches needed for the program to compile and install on FreeBSD. This directory may also contain other files used to build the port.
- **pkg-descr**: provides a more detailed description of the program.
- **pkg-plist**: a list of all the files that will be installed by the port. It also tells the ports system which files to remove upon deinstallation.

Some ports include `pkg-message` or other files to handle special situations. For more details on these files, and on ports in general, refer to the FreeBSD Porter's Handbook.

The port does not include the actual source code, also known as a `distfile`. The extract portion of building a port will automatically save the downloaded source to `/usr/ports/distfiles`.

4.5.1. Installing the Ports Collection

Before an application can be compiled using a port, the Ports Collection must first be installed. If it was not installed during the installation of FreeBSD, use the following method to install it:

Procedure: Git Method

If more control over the ports tree is needed or if local changes need to be maintained, or if running FreeBSD-CURRENT, Git can be used to obtain the Ports Collection. Refer to the Git Primer for a detailed description of Git.

We add `--depth 1` to the git command line to clone the tree without obtaining the commit history, which saves time and is acceptable for most users. When maintaining custom changes to the ports tree, or need the history for any reason, omit the `--depth 1` argument below.

1. Git must be installed before it can be used to check out the ports tree. If a copy of the ports tree is already present, install Git like this:

   ```
   # cd /usr/ports/devel/git
   # make install clean
   ```

   If the ports tree is not available, or pkg is being used to manage packages, Git can be installed as a package:

   ```
   # pkg install git
   ```

2. Check out a copy of the HEAD branch of the ports tree:

   ```
   # git clone --depth 1 https://git.FreeBSD.org/ports.git /usr/ports
   ```

3. Or, check out a copy of a quarterly branch:

   ```
   # git clone --depth 1 https://git.FreeBSD.org/ports.git -b 2023Q1 /usr/ports
   ```

4. As needed, update `/usr/ports` after the initial Git checkout:

   ```
   # git -C /usr/ports pull
   ```

5. As needed, switch `/usr/ports` to a different quarterly branch:

   ```
   # git -C /usr/ports fetch --depth 1 origin 2023Q2:2023Q2
   # git -C /usr/ports switch 2023Q2
   ```

4.5.2. Installing Ports

This section provides basic instructions on using the Ports Collection to install or remove software. The detailed description of available `make` targets and environment variables is available in ports(7).

Before compiling any port, be sure to update the Ports Collection as described in the previous section. Since the installation of any third-party software can introduce security vulnerabilities, it is recommended to first check https://vuxml.freebsd.org/ for known security issues related to the port. Alternatively, run `pkg audit -F` before installing a new port. This command can be configured to automatically perform a security audit and an update of the vulnerability database during the daily security system check. For more information, refer to pkg-audit(8) and periodic(8).

Using the Ports Collection assumes a working Internet connection. It also requires superuser privilege.

To compile and install the port, change to the directory of the port to be installed, then type `make install` at the prompt. Messages will indicate the progress:

```
# cd /usr/ports/sysutils/lsof
# make install
>> lsof_4.88D.freebsd.tar.gz doesn't seem to exist in /usr/ports/distfiles/.
>> Attempting to fetch from ftp://lsof.itap.purdue.edu/pub/tools/unix/lsof/.
===>  Extracting for lsof-4.88
...
[extraction output snipped]
```

```
...
>> Checksum OK for lsof_4.88D.freebsd.tar.gz.
===>  Patching for lsof-4.88.d,8
===>  Applying FreeBSD patches for lsof-4.88.d,8
===>  Configuring for lsof-4.88.d,8
...
[configure output snipped]
...
===>  Building for lsof-4.88.d,8
...
[compilation output snipped]
...

===>  Installing for lsof-4.88.d,8
...
[installation output snipped]
...
===>   Generating temporary packing list
===>   Compressing manual pages for lsof-4.88.d,8
===>   Registering installation for lsof-4.88.d,8
===>  SECURITY NOTE:
      This port has installed the following binaries which execute with
      increased privileges.
/usr/local/sbin/lsof
#
```

Since `lsof` is a program that runs with increased privileges, a security warning is displayed as it is installed. Once the installation is complete, the prompt will be returned.

Some shells keep a cache of the commands that are available in the directories listed in the `PATH` environment variable, to speed up lookup operations for the executable file of these commands. Users of the `tcsh` shell should type `rehash` so that a newly installed command can be used without specifying its full path. Use `hash -r` instead for the `sh` shell. Refer to the documentation for the shell for more information.

During installation, a working subdirectory is created which contains all the temporary files used during compilation. Removing this directory saves disk space and minimizes the chance of problems later when upgrading to the newer version of the port:

```
# make clean
===>  Cleaning for lsof-88.d,8
#
```

To save this extra step, instead use `make install clean` when compiling the port.

Customizing Ports Installation

Some ports provide build options which can be used to enable or disable application components, provide security options, or allow for other customizations. Examples include www/firefox and security/gpgme. If the port depends upon other ports which have configurable options, it may pause several times for user interaction as the default behavior is to prompt the user to select options from a menu. To avoid this and do all of the configuration in one batch, run `make config-recursive` within the port skeleton. Then, run `make install [clean]` to compile and install the port.

> When using `config-recursive`, the list of ports to configure are gathered by the `all-depends-list` target. It is recommended to run `make config-recursive` until all dependent ports options have been defined, and ports options screens no longer appear, to be certain that all dependency options have been configured.

There are several ways to revisit a port's build options menu in order to add, remove, or change these options after a port has been built. One method is to `cd` into the directory containing the port and type `make config`. Another option is to use `make showconfig`. Another option is to execute `make rmconfig` which will remove all selected options and allows to start over. All of these options, and others, are explained in great detail in ports(7).

The ports system uses fetch(1) to download the source files, which supports various environment variables. The `FTP_PASSIVE_MODE`, `FTP_PROXY`, and `FTP_PASSWORD` variables may need to be set if the FreeBSD system is behind a firewall or FTP/HTTP proxy. See fetch(3) for the complete list of supported variables.

For users who cannot be connected to the Internet all the time, `make fetch` can be run within `/usr/ports`, to fetch all distfiles, or within a category, such as `/usr/ports/net`, or within the specific port skeleton. Note that if a port has any dependencies, running this command in a category or ports skeleton will *not* fetch the distfiles of ports from another category. Instead, use `make fetch-recursive` to also fetch the distfiles for all the dependencies of a port.

In rare cases, such as when an organization has a local distfiles repository, the `MASTER_SITES` variable can be used to override the download locations specified in the `Makefile`. When using, specify the alternate location:

```
# cd /usr/ports/directory
# make MASTER_SITE_OVERRIDE= \
ftp://ftp.organization.org/pub/FreeBSD/ports/distfiles/ fetch
```

The `WRKDIRPREFIX` and `PREFIX` variables can override the default working and target directories. For example:

```
# make WRKDIRPREFIX=/usr/home/example/ports install
```

will compile the port in `/usr/home/example/ports` and install everything under `/usr/local`.

```
# make PREFIX=/usr/home/example/local install
```

will compile the port in `/usr/ports` and install it in `/usr/home/example/local`. And:

```
# make WRKDIRPREFIX=../ports PREFIX=../local install
```

will combine the two.

These can also be set as environmental variables. Refer to the manual page for the shell for instructions on how to set an environmental variable.

4.5.3. Removing Installed Ports

Installed ports can be uninstalled using `pkg delete`. Examples for using this command can be found in the pkg-delete(8) manual page.

Alternately, `make deinstall` can be run in the port's directory:

```
# cd /usr/ports/sysutils/lsof
# make deinstall
===>  Deinstalling for sysutils/lsof
===>   Deinstalling
Deinstallation has been requested for the following 1 packages:

    lsof-4.88.d,8

The deinstallation will free 229 kB
[1/1] Deleting lsof-4.88.d,8... done
```

It is recommended to read the messages as the port is uninstalled. If the port has any applications that depend upon it, this information will be displayed but the uninstallation will proceed. In such cases, it may be better to reinstall the application in order to prevent broken dependencies.

4.5.4. Upgrading Ports

Over time, newer versions of software become available in the Ports Collection. This section describes how to determine which software can be upgraded and how to perform the upgrade.

To determine if newer versions of installed ports are available, ensure that the latest version of the ports tree is installed, using the updating command described in Git Method. The following command will list the installed ports which are out of date:

```
# pkg version -l "<"
```

Before attempting an upgrade, read `/usr/ports/UPDATING` from the top of the file to the date closest to the last time ports were upgraded or the system was installed. This file describes various issues and additional steps users may encounter and need to perform when updating a port, including such things as file format changes, changes in locations of configuration files, or any incompatibilities with previous versions. Make note of any instructions which match any of the ports that need upgrading and follow these instructions when performing the upgrade.

Tools to Upgrade and Manage Ports

The Ports Collection contains several utilities to perform the actual upgrade. Each has its strengths and weaknesses.

Historically, most installations used either Portmaster or Portupgrade. Synth is a newer alternative.

> The choice of which tool is best for a particular system is up to the system administrator. It is recommended practice to back up data before using any of these tools.

Upgrading Ports Using Portmaster

ports-mgmt/portmaster is a very small utility for upgrading installed ports. It is designed to use the tools installed with the FreeBSD base system without depending on other ports or databases. To install this utility as a port:

```
# cd /usr/ports/ports-mgmt/portmaster
# make install clean
```

Portmaster defines four categories of ports:

- Root port: has no dependencies and is not a dependency of any other ports.
- Trunk port: has no dependencies, but other ports depend upon it.
- Branch port: has dependencies and other ports depend upon it.
- Leaf port: has dependencies but no other ports depend upon it.

To list these categories and search for updates:

```
# portmaster -L
===>>> Root ports (No dependencies, not depended on)
===>>> ispell-3.2.06_18
===>>> screen-4.0.3
        ===>>> New version available: screen-4.0.3_1
===>>> tcpflow-0.21_1
===>>> 7 root ports
...
===>>> Branch ports (Have dependencies, are depended on)
===>>> apache22-2.2.3
        ===>>> New version available: apache22-2.2.8
...
===>>> Leaf ports (Have dependencies, not depended on)
===>>> automake-1.9.6_2
===>>> bash-3.1.17
        ===>>> New version available: bash-3.2.33
```

```
...
===>>> 32 leaf ports

===>>> 137 total installed ports
        ===>>> 83 have new versions available
```

This command is used to upgrade all outdated ports:

```
# portmaster -a
```

By default, Portmaster makes a backup package before deleting the existing port. If the installation of the new version is successful, Portmaster deletes the backup. Using `-b` instructs Portmaster not to automatically delete the backup. Adding `-i` starts Portmaster in interactive mode, prompting for confirmation before upgrading each port. Many other options are available. Read through the manual page for portmaster(8) for details regarding their usage.

If errors are encountered during the upgrade process, add `-f` to upgrade and rebuild all ports:

```
# portmaster -af
```

Portmaster can also be used to install new ports on the system, upgrading all dependencies before building and installing the new port. To use this function, specify the location of the port in the Ports Collection:

```
# portmaster shells/bash
```

More information about ports-mgmt/portmaster may be found in its `pkg-descr`.

Upgrading Ports Using Portupgrade

Portupgrade is deprecated and will be removed in the near future.

ports-mgmt/portupgrade is another utility that can be used to upgrade ports. It installs a suite of applications which can be used to manage ports. However, it is dependent upon Ruby. To install the port:

```
# cd /usr/ports/ports-mgmt/portupgrade
# make install clean
```

Before performing an upgrade using this utility, it is recommended to scan the list of installed ports using `pkgdb -F` and to fix all the inconsistencies it reports.

To upgrade all the outdated ports installed on the system, use `portupgrade -a`. Alternately, include `-i` to be asked for confirmation of every individual upgrade:

```
# portupgrade -ai
```

To upgrade only a specified application instead of all available ports, use `portupgrade pkgname`. It is very important to include `-R` to first upgrade all the ports required by the given application:

```
# portupgrade -R firefox
```

If `-P` is included, Portupgrade searches for available packages in the local directories listed in `PKG_PATH`. If none are available locally, it then fetches packages from a remote site. If packages can not be found locally or fetched remotely, Portupgrade will use ports. To avoid using ports entirely, specify `-PP`. This last set of options tells Portupgrade to abort if no packages are available:

```
# portupgrade -PP gnome3
```

To just fetch the port distfiles, or packages, if `-P` is specified, without building or installing anything, use `-F`. For further information on all of the available switches, refer to the manual page for `portupgrade`.

More information about ports-mgmt/portupgrade may be found in its `pkg-descr`.

4.5.5. Ports and Disk Space

Using the Ports Collection will use up disk space over time. After building and installing a port, running `make clean` within the ports skeleton will clean up the temporary `work` directory. If Portmaster is used to install a port, it will automatically remove this directory unless `-K` is specified. If Portupgrade is installed, this command will remove all `work` directories found within the local copy of the Ports Collection:

```
# portsclean -C
```

In addition, outdated source distribution files accumulate in `/usr/ports/distfiles` over time. To use Portupgrade to delete all the distfiles that are no longer referenced by any ports:

```
# portsclean -D
```

Portupgrade can remove all distfiles not referenced by any port currently installed on the system:

```
# portsclean -DD
```

If Portmaster is installed, use:

```
# portmaster --clean-distfiles
```

By default, this command is interactive and prompts the user to confirm if a distfile should be deleted.

In addition to these commands, ports-mgmt/pkg_cutleaves automates the task of removing installed ports that are no longer needed.

4.6. Building Packages with poudriere

poudriere is a `BSD`-licensed utility for creating and testing FreeBSD packages. It uses FreeBSD jails to set up isolated compilation environments. These jails can be used to build packages for versions of FreeBSD that are different from the system on which it is installed, and also to build packages for i386 if the host is an amd64 system. Once the packages are built, they are in a layout identical to the official mirrors. These packages are usable by pkg(8) and other package management tools.

poudriere is installed using the ports-mgmt/poudriere package or port. The installation includes a sample configuration file `/usr/local/etc/poudriere.conf.sample`. Copy this file to `/usr/local/etc/poudriere.conf`. Edit the copied file to suit the local configuration.

While `ZFS` is not required on the system running poudriere, it is beneficial. When `ZFS` is used, `ZPOOL` must be specified in `/usr/local/etc/poudriere.conf` and `FREEBSD_HOST` should be set to a nearby mirror. Defining `CCACHE_DIR` enables the use of devel/ccache to cache compilation and reduce build times for frequently-compiled code. It may be convenient to put poudriere datasets in an isolated tree mounted at `/poudriere`. Defaults for the other configuration values are adequate.

The number of processor cores detected is used to define how many builds will run in parallel. Supply enough virtual memory, either with `RAM` or swap space. If virtual memory runs out, the compilation jails will stop and be torn down, resulting in weird error messages.

4.6.1. Initialize Jails and Port Trees

After configuration, initialize poudriere so that it installs a jail with the required FreeBSD tree and a ports tree. Specify a name for the jail using `-j` and the FreeBSD version with `-v`. On systems running FreeBSD/amd64, the architecture can be set with `-a` to either `i386` or `amd64`. The default is the architecture shown by `uname`.

```
# poudriere jail -c -j 13amd64 -v 13.1-RELEASE
[00:00:00] Creating 13amd64 fs at /poudriere/jails/13amd64... done
[00:00:00] Using pre-distributed MANIFEST for FreeBSD 13.1-RELEASE amd64
[00:00:00] Fetching base for FreeBSD 13.1-RELEASE amd64
/poudriere/jails/13amd64/fromftp/base.txz             125 MB 4110 kBps    31s
[00:00:33] Extracting base... done
[00:00:54] Fetching src for FreeBSD 13.1-RELEASE amd64
/poudriere/jails/13amd64/fromftp/src.txz              154 MB 4178 kBps    38s
[00:01:33] Extracting src... done
[00:02:31] Fetching lib32 for FreeBSD 13.1-RELEASE amd64
/poudriere/jails/13amd64/fromftp/lib32.txz             24 MB 3969 kBps    06s
```

```
[00:02:38] Extracting lib32... done
[00:02:42] Cleaning up... done
[00:02:42] Recording filesystem state for clean... done
[00:02:42] Upgrading using ftp
/etc/resolv.conf -> /poudriere/jails/13amd64/etc/resolv.conf
Looking up update.FreeBSD.org mirrors... 3 mirrors found.
Fetching public key from update4.freebsd.org... done.
Fetching metadata signature for 13.1-RELEASE from update4.freebsd.org... done.
Fetching metadata index... done.
Fetching 2 metadata files... done.
Inspecting system... done.
Preparing to download files... done.
Fetching 124
patches.....10....20....30....40....50....60....70....80....90....100....110....120..
done.
Applying patches... done.
Fetching 6 files... done.
The following files will be added as part of updating to
13.1-RELEASE-p1:
/usr/src/contrib/unbound/.github
/usr/src/contrib/unbound/.github/FUNDING.yml
/usr/src/contrib/unbound/contrib/drop2rpz
/usr/src/contrib/unbound/contrib/unbound_portable.service.in
/usr/src/contrib/unbound/services/rpz.c
/usr/src/contrib/unbound/services/rpz.h
/usr/src/lib/libc/tests/gen/spawnp_enoexec.sh
The following files will be updated as part of updating to
13.1-RELEASE-p1:
[…]
Installing updates...Scanning //usr/share/certs/blacklisted for certificates...
Scanning //usr/share/certs/trusted for certificates...
 done.
13.1-RELEASE-p1
[00:04:06] Recording filesystem state for clean... done
[00:04:07] Jail 13amd64 13.1-RELEASE-p1 amd64 is ready to be used
```

```
# poudriere ports -c -p local -m git+https
[00:00:00] Creating local fs at /poudriere/ports/local... done
[00:00:00] Checking out the ports tree... done
```

On a single computer, poudriere can build ports with multiple configurations, in multiple jails, and from different port trees. Custom configurations for these combinations are called *sets*. See the CUSTOMIZATION section of poudriere(8) for details after ports-mgmt/poudriere or ports-mgmt/poudriere-devel is installed.

The basic configuration shown here puts a single jail-, port-, and set-specific `make.conf` in `/usr/local/etc/poudriere.d`. The filename in this example is created by combining the jail name, port name, and set name: `13amd64-local-workstation-make.conf`. The system `make.conf` and this new file are combined at build time to create the `make.conf` used by the build jail.

Packages to be built are entered in `13amd64-local-workstation-pkglist` (ports with FLAVORS can be defined with @FLAVOR):

```
editors/emacs
devel/git
devel/php-composer2@php82
ports-mgmt/pkg
...
```

Options and dependencies for the specified ports are configured:

```
# poudriere options -j 13amd64 -p local -z workstation -f 13amd64-local-workstation-
pkglist
```

Finally, packages are built and a package repository is created:

```
# poudriere bulk -j 13amd64 -p local -z workstation -f 13amd64-local-workstation-
pkglist
```

While running, pressing `Ctrl` + `t` displays the current state of the build. poudriere also builds files in `/poudriere/logs/bulk/jailname` that can be used with a web server to display build information.

After completion, the new packages are now available for installation from the poudriere repository.

For more information on using poudriere, see poudriere(8) and the main web site, https://github.com/freebsd/poudriere/wiki.

4.6.2. Configuring pkg Clients to Use a poudriere Repository

While it is possible to use both a custom repository along side of the official repository, sometimes it is useful to disable the official repository. This is done by creating a configuration file that overrides and disables the official configuration file. Create `/usr/local/etc/pkg/repos/FreeBSD.conf` that contains the following:

```
FreeBSD: {
    enabled: no
}
```

Usually it is easiest to serve a poudriere repository to the client machines via HTTP. Set up a webserver to serve up the package directory, for instance: `/usr/local/poudriere/data/packages/13amd64`, where `13amd64` is the name of the build.

If the URL to the package repository is: `https://pkg.example.com/13amd64`, then the repository configuration file in `/usr/local/etc/pkg/repos/custom.conf` would look like:

```
custom: {
    url: "https://pkg.example.com/13amd64",
    enabled: yes,
}
```

If exposing the package repository to the internet is not desired, the `file://` protocol can be used to point to the repository directly:

```
custom: {
    url: "file:///usr/local/poudriere/data/packages/11amd64",
    enabled: yes,
}
```

4.7. Post-Installation Considerations

Regardless of whether the software was installed from a binary package or port, most third-party applications require some level of configuration after installation. The following commands and locations can be used to help determine what was installed with the application.

- Most applications install at least one default configuration file in `/usr/local/etc`. In cases where an application has a large number of configuration files, a subdirectory will be created to hold them. Often, sample configuration files are installed which end with a suffix such as `.sample`. The configuration files should be reviewed and possibly edited to meet the system's needs. To edit a sample file, first copy it without the `.sample` extension.
- Applications which provide documentation will install it into `/usr/local/share/doc` and many applications also install manual pages. This documentation should be consulted before continuing.
- Some applications run services which must be added to `/etc/rc.conf` before starting the application. These applications usually install a startup script in `/usr/local/etc/rc.d`. See Starting Services for more information.

> By design, applications do not run their startup script upon installation, nor do they run their stop script upon deinstallation or upgrade. This decision is left to the individual system administrator.

- Users of csh(1) should run `rehash` to rebuild the known binary list in the shells `PATH`.
- Use `pkg info` to determine which files, man pages, and binaries were installed with the application.

4.8. Dealing with Broken Ports

When a port does not build or install, try the following:

1. Search to see if there is a fix pending for the port in the Problem Report database. If so,

implementing the proposed fix may fix the issue.

2. Ask the maintainer of the port for help. Type `make maintainer` in the ports skeleton or read the port's `Makefile` to find the maintainer's email address. Remember to include the output leading up to the error in the email to the maintainer.

   > Some ports are not maintained by an individual but instead by a group maintainer represented by a mailing list. Many, but not all, of these addresses look like freebsd-listname@FreeBSD.org. Please take this into account when sending an email.
   >
   > In particular, ports maintained by ports@FreeBSD.org are not maintained by a specific individual. Instead, any fixes and support come from the general community who subscribe to that mailing list. More volunteers are always needed!

   If there is no response to the email, use Bugzilla to submit a bug report using the instructions in Writing FreeBSD Problem Reports.

3. Fix it! The Porter's Handbook includes detailed information on the ports infrastructure, enabling the fixing of occasional broken ports or the submission of new ones. Install the package instead of the port using the instructions in Using pkg for Binary Package Management.

Chapter 5. The X Window System

5.1. Synopsis

An installation of FreeBSD using bsdinstall(8) does not automatically install a graphical user interface. This chapter describes how to install and configure the Xorg(1) server, which provides the open source X Window System (colloquially X11) used to provide a graphical environment.

Before reading this chapter:

- Know how to install additional third-party software as described in Installing Applications: Packages and Ports.

Read this chapter to learn:

- How to select and install drivers for the graphics processor (GPU).
- The various components of the X Window System, and how they interoperate.
- How to install and configure the X.org server.
- How to install fonts for the X Window System.

5.2. Graphics Drivers

***abstract**: Identify the GPU, the port providing a driver for it, install it, then enable it to run at subsequent boot with sysrc(8).*

Before FreeBSD can render a graphical environment, it needs a kernel module to drive the graphics processor. Graphics drivers are a fast-moving, cross-platform target, which is why this is developed and distributed separately from the FreeBSD base system.

The following table shows the different graphics processors supported by FreeBSD, their corresponding module, and which port provides it:

Table 8. Supported Graphics Devices

Type	License	Module	Port
Intel®	Open Source	`i915kms`	graphics/drm-kmod
AMD®	Open Source	`amdgpu` or `radeonkms`	graphics/drm-kmod
NVIDIA®	Proprietary	`nvidia-drm`, `nvidia-modeset`, or `nvidia`	graphics/nvidia-drm-kmod or x11/nvidia-driver
System Console Framebuffer	Open Source	`scfb`	x11-drivers/xf86-video-scfb
VESA BIOS Extension	Open Source	`vesa`	x11-drivers/xf86-video-vesa

Type	License	Module	Port
VirtualBox®	Open Source	`vboxvideo`	emulators/virtualbox-ose-additions
VMware®	Open Source	`vmwgfx`	x11-drivers/xf86-video-vmware

There are several generations of driver technologies supported.

- Direct Rendering drivers allowing PRIME offloading. PRIME allows for multiple providers of graphics processing to coexist. PRIME is described further in Graphics Configuration.
- Kernel Modesetting (KMS) This allows the driver to directly specify the display mode. This is required to support suspend and resume when using the vt(4) console driver.
- User Modesetting The oldest class of drivers is still supported, however they may only be used with sc(4) console and older versions of the Xorg(1) graphical environment.

The following command can be used to identify which graphics processor is installed in the system:

```
% pciconf -lv | grep -B3 display
```

The output should be similar to the following:

```
vgapci1@pci0:0:2:0:     class=0x030000 rev=0x0c hdr=0x00 vendor=0x8086 device=0x46a6
subvendor=0x1028 subdevice=0x0b29
    vendor     = 'Intel Corporation'
    device     = 'Alder Lake-P GT2 [Iris Xe Graphics]'
    class      = display
```

Detailed instructions on installing and enabling these drivers are in the subsequent subsections.

If the graphics processor is not supported by Intel®, AMD®, or NVIDIA® drivers, then SCFB or VESA modules should be used. SCFB module must be used when booting in UEFI mode. VESA module must be used when booting in BIOS mode.

This command can be used to check the booting mode:

```
% sysctl machdep.bootmethod
```

The output should be similar to the following:

```
machdep.bootmethod: UEFI
```

5.2.1. Intel® Graphics

The graphics/drm-kmod package indirectly provides a range of kernel modules for use with Intel® Graphics. Recent versions of these modules can be used in conjunction with other graphics processors in PRIME with no special configuration.

The Intel® Graphics driver can be installed by executing the following command:

```
# pkg install drm-kmod
```

Then add the module to /etc/rc.conf file, by executing the following command:

```
# sysrc kld_list+=i915kms
```

5.2.2. AMD® Graphics

The graphics/drm-kmod package indirectly provides kernel modules for a range of AMD® Graphics processors. The modules `amdgpu` or `radeonkms` can be used depending the generation of the hardware. The FreeBSD project maintains a AMD graphics support matrix showing support levels and to determine which driver must be used.

The AMD® Graphics drivers can be installed by executing the following command:

```
# pkg install drm-kmod
```

Enable the current module by adding it to the /etc/rc.conf file, executing the following command:

```
# sysrc kld_list+=amdgpu
```

For older graphics (pre-HD7000/Tahiti), instead enable the legacy module by adding it to the /etc/rc.conf file, executing the following command:

```
# sysrc kld_list+=radeonkms
```

5.2.3. NVIDIA® Graphics

NVIDIA® produces standalone or discrete graphics processors, and provides a proprietary driver for FreeBSD. The FreeBSD Ports Collection provides over a decade of drivers for supporting generations of NVIDIA graphics.

Administrators should install the latest driver supported by their hardware.

The following table shows the port containing the driver, the kernel module recommended for loading, and a link to the list of hardware supported by that driver:

Table 9. Supported versions of NVIDIA® Graphics drivers

Port	Module	Supported hardware
graphics/nvidia-drm-kmod	`nvidia-drm` or `nvidia-modeset`	supported hardware
x11/nvidia-driver-470	`nvidia-modeset`	supported hardware
x11/nvidia-driver-390 or x11/nvidia-secondary-driver-390	`nvidia-modeset`	supported hardware
x11/nvidia-driver-340	`nvidia`	supported hardware
x11/nvidia-driver-304	`nvidia`	supported hardware

The latest NVIDIA® Graphics driver can be installed by running the following command:

```
# pkg install nvidia-drm-kmod
```

To enable the driver, add the module to /etc/rc.conf file, by executing the following command:

```
# sysrc kld_list+=nvidia-drm
```

This is the direct rendering KMS driver.

Kernel modesetting is the option to set the graphics mode in the kernel. Enable it for subsequent boots with the following loader.conf(5) tunable:

```
hw.nvidiadrm.modeset="1"
```

Both PRIME and Wayland require kernel modesetting.

Prior versions of the driver do not support Direct Rendering. Instead use the modesetting module, by executing the following command:

```
# sysrc kld_list+=nvidia-modeset
```

If requiring Nvidia drivers prior to 390, note that they do not support kernel modesetting, and thus they must be used with the legacy sc(4) console driver, and a x11/xorg-server version prior to 1.20.

Enable them in /etc/rc.conf with the following command:

```
# sysrc kld_list+=nvidia
```

Some graphic cards will need special firmware that needs to be downloaded for the device to work properly. The fwget(8) utility can download and install the necessary files under /boot/firmware.

fwget(8) will scan the 'PCI' devices and download the firmware for those devices that need it.

5.3. X Window System Overview

The X Window System is the heritage graphical stack for UNIX® platforms, supporting the latest technologies while maintaining support for generations of applications. Applications, including the components of the desktop, are hosted by the Xorg(1) server. This system is network aware and its various components can interoperate across networks.

5.4. Installing The X.org Server

***abstract**: The x11/xorg server must be installed to host the desktop. Users must be added to the `video` group to use it.*

Once a graphics driver is installed and enabled, the X.org server can be installed as a meta-package, or compiled locally with the ports tree.

The full meta-package can be installed quickly but with fewer options for customization:

```
# pkg install xorg
```

This installation results in the complete X Window System being installed, including a traditional window manager, twm(1), and surrounding traditional desktop suite. Most users will want to install and configure a contemporary desktop of their choice.

The current user must be a member of the `video` group to run a graphical environment. To add a user to the `video` group, execute the following command:

```
# pw groupmod video -m username
```

To run the X Window System, use startx(1) from x11/xinit, or install and configure a display manager to start a graphical login on boot.

A smaller version of the X Window System suitable for experienced users is available in x11/xorg-minimal. Most of the documents, libraries, and applications will not be installed. Some applications require these additional components to function.

5.5. X.org Configuration

***abstract**: If the defaults for the monitor or input devices, are not satisfactory, desktops include GUIs for configuring them, or they can be configured manually.*

The X.org server supports most common graphics processors, monitors, and input devices. First, try the defaults. This subsection provides an overview of their configuration.

5.5.1. X.org Configuration Files

Historically, the X.org server was configured with files in /usr/local/etc/X11/. This is still supported for edge cases, but conflicts with dynamic autoconfiguration.

Do not create configuration for the X.org server in xorg.conf or run `Xorg -configure` unless automatic configuration fails.

X.org server looks in several directories for configuration files. /usr/local/etc/X11/ is the **recommended** directory for these files on FreeBSD. Using this directory helps keep application files separate from operating system files.

It is easier to use multiple files that each configure a specific setting than the traditional single xorg.conf. These files are stored in the /usr/local/etc/X11/xorg.conf.d/ subdirectory.

5.5.2. Graphics Configuration

Direct rendering provides the ability to seamlessly use a discrete graphics processor (dGPU) alongside an integrated graphics processor (iGPU), called PRIME. The drivers will automatically offload intensive tasks to the dGPU when required, and power it down when able.

To launch applications on the more powerful GPU in PRIME, use the `DRI_PRIME=1` environment variable.

If multiple graphics drivers are conflicting, the driver for the graphics processor can be specified in the /usr/local/etc/X11/xorg.conf.d/ directory.

To configure the Intel® driver in a configuration file:

Example 14. Select Intel® Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-intel.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "intel"
EndSection
```

To configure the AMD® driver in a configuration file:

Example 15. Select AMD® Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-radeon.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "radeon"
EndSection
```

To configure the NVIDIA® driver in a configuration file:

Example 16. Select NVIDIA® Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-nvidia.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "nvidia"
EndSection
```

x11/nvidia-xconfig can also be used to perform basic control over configuration options available in the NVIDIA driver.

To configure the SCFB driver in a configuration file:

Example 17. Select SCFB Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-scfb.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "scfb"
EndSection
```

To configure the VESA driver in a configuration file:

Example 18. Select VESA Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-vesa.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "vesa"
EndSection
```

To configure multiple graphics processors, the `BusID` can be added. A list of graphics processor bus `ID`s can be displayed by executing:

```
% pciconf -lv | grep -B3 display
```

The output should be similar to the following:

```
vgapci0@pci0:0:2:0:     class=0x030000 rev=0x0c hdr=0x00 vendor=0x8086 device=0x46a6
subvendor=0x1028 subdevice=0x0b29
    vendor     = 'Intel Corporation'
    device     = 'Alder Lake-P GT2 [Iris Xe Graphics]'
    class      = display
--
vgapci0@pci0:1:0:0:     class=0x030200 rev=0xa1 hdr=0x00 vendor=0x10de device=0x25b9
subvendor=0x1028 subdevice=0x0b29
    vendor     = 'NVIDIA Corporation'
    device     = 'GA107GLM [RTX A1000 Laptop GPU]'
    class      = display
```

Example 19. Select Intel® Graphics Driver and NVIDIA® Graphics Driver in a File

/usr/local/etc/X11/xorg.conf.d/20-drivers.conf

```
Section "Device"
    Identifier "Card0"
    Driver     "intel"
    BusID     "pci0:0:2:0"
EndSection

Section "Device"
    Identifier "Card1"
    Driver     "nvidia"
    BusID     "pci0:1:0:0"
EndSection
```

5.5.3. Monitor Configuration

Almost all monitors support the Extended Display Identification Data standard (`EDID`). X.org uses `EDID` to communicate with the monitor and detect the supported resolutions and refresh rates. Then it selects the most appropriate combination of settings to use with that monitor.

Other resolutions supported by the monitor can be selected atomically after the X server has been started with xrandr(1), or in the X.org server configuration files.

Using RandR (Resize and Rotate)

Run xrandr(1) in an X session without any parameters to see a list of video outputs and detected monitor modes:

```
% xrandr
```

The output should be similar to the following:

```
Screen 0: minimum 320 x 200, current 2560 x 960, maximum 8192 x 8192
LVDS-1 connected 1280x800+0+0 (normal left inverted right x axis y axis) 261mm x 163mm
   1280x800      59.99*+  59.81    59.91    50.00
   1280x720      59.86    59.74
   1024x768      60.00
   1024x576      59.90    59.82
   960x540       59.63    59.82
   800x600       60.32    56.25
   864x486       59.92    59.57
   640x480       59.94
   720x405       59.51    58.99
   640x360       59.84    59.32
VGA-1 connected primary 1280x960+1280+0 (normal left inverted right x axis y axis)
410mm x 257mm
   1280x1024     75.02    60.02
   1440x900      74.98    60.07
   1280x960      60.00*
   1280x800      74.93    59.81
   1152x864      75.00
   1024x768      75.03    70.07    60.00
   832x624       74.55
   800x600       72.19    75.00    60.32    56.25
   640x480       75.00    72.81    66.67    59.94
   720x400       70.08
HDMI-1 disconnected (normal left inverted right x axis y axis)
DP-1 disconnected (normal left inverted right x axis y axis)
HDMI-2 disconnected (normal left inverted right x axis y axis)
DP-2 disconnected (normal left inverted right x axis y axis)
DP-3 disconnected (normal left inverted right x axis y axis)
```

This shows that the `VGA-1` output is being used to display a screen resolution of 1280x960 pixels at a refresh rate of about 60 Hz. The `LVDS-1` is being used as a secondary monitor to display a screen resolution of 1280x800 pixels at a refresh rate of about 60 Hz. Monitors are not attached to the `HDMI-1`, `HDMI-2`, `DP-1`, `DP-2`, and `DP-3` connectors.

Any of the other display modes can be selected with xrandr(1). For example, to switch to 1280x1024 at 60 Hz:

```
% xrandr --output LVDS-1 --mode 1280x720 --rate 60
```

Oftentimes, a black screen upon starting X can be fixed by adding an `xrandr --auto` step to the initialization process.

Using the X.org Configuration Files

The monitor configuration can also be set in a configuration file.

To set a screen resolution of 1024x768 in a configuration file:

Example 20. Set Screen Resolution in a File

/usr/local/etc/X11/xorg.conf.d/10-monitor.conf

```
Section "Screen"
    Identifier "Screen0"
    Device     "Card0"
    SubSection "Display"
    Modes      "1024x768"
    EndSubSection
EndSection
```

5.5.4. Input Configuration

The Xorg(1) server provides the x11/libinput library, a cross-platform effort to support all touch, pointing, and keyboard devices under a unified library. Unless specified otherwise, this is loaded automatically.

Individual device settings for libinput(4) can be tuned in the desktop's GUI, or manually with x11/xinput and x11/setxkbmap.

Alternatively, there are older, lightweight, individual drivers for specific input devices available in the x11-drivers category named x11/xf86-input-[foo]. This approach requires manual configuration of the X.org server. Both are described in this subsection.

Using Atomic Input Configuration

Devices supported by libinput(4) can be configured with graphical utilities included with the desktop of choice, or manually and atomically at runtime with x11/xinput and x11/setxkbmap.

To ask libinput(4) what devices it's currently attached to, run xinput(1) with no arguments:

```
$ xinput
```

Its output should be similar to the following:

```
⎡ Virtual core pointer                      id=2    [master pointer  (3)]
⎜   ↳ Virtual core XTEST pointer                id=4    [slave  pointer  (2)]
⎜   ↳ System mouse                              id=7    [slave  pointer  (2)]
⎜   ↳ VEN_0488:00 0488:1031 Mouse               id=11   [slave  pointer  (2)]
⎜   ↳ VEN_0488:00 0488:1031 TouchPad            id=12   [slave  pointer  (2)]
⎣ Virtual core keyboard                     id=3    [master keyboard (2)]
    ↳ Virtual core XTEST keyboard               id=5    [slave  keyboard (3)]
    ↳ System keyboard multiplexer               id=6    [slave  keyboard (3)]
    ↳ Power Button                              id=8    [slave  keyboard (3)]
    ↳ Sleep Button                              id=9    [slave  keyboard (3)]
    ↳ AT keyboard                               id=10   [slave  keyboard (3)]
```

All settings supported by these devices are provided as properties, which can be listed and set atomically. Pointing devices have many configurable properties, keyboards usually need none.

To customize the keyboard, take a look at setxkbmap(1).

Once satisfied with the configuration, simply add the lines to the X initialization script such as ~/.Xsession or ~/.xinitrc.

Using X.org Configuration Files

> Some desktop environments (such as KDE Plasma) provide a graphical UI for setting these parameters. Check if this is the case before resorting to manual configuration editing.

For example, to manually configure the X.org server for the keyboard layout:

Example 21. Setting a Keyboard Layout

/usr/local/etc/X11/xorg.conf.d/00-keyboard.conf

```
Section "InputClass"
        Identifier "Keyboard1"
        MatchIsKeyboard "on"
        Option "XkbLayout" "es, fr"
        Option "XkbModel" "pc104"
        Option "XkbVariant" ",qwerty"
        Option "XkbOptions" "grp:win_space_toggle"
EndSection
```

5.6. Using Fonts in the X Window System

abstract: *Additional fonts can be installed from the x11-fonts category or placed in ~/.fonts. They are available immediately to modern applications. Configuration for older applications is available and described as well.*

The X Window System provides the X FreeType interface library (Xft(3)) to render vector or outline fonts, as well as the traditional X Logical Font Description system maintaining compatibility with generations of applications and fonts.

There are primarily two types of fonts users will be interested in:

- OpenType fonts or TrueType® fonts are for displaying on a screen.
- Adobe® PostScript® Type 1 fonts are for printing to paper.

These are both vector or outline fonts, there are also bitmap fonts.

The FreeBSD Ports Collection includes a wide and growing catalog of free, high quality fonts available for installation in the x11-fonts category.

System-wide font packages installed from the ports collection live in `/usr/local/share/fonts/`. Fonts for a single user can be placed in `~/.fonts/`, or `~/.local/share/fonts/`.

Fonts in either directory or subdirectories will be available for immediate use when the font information cache is rebuilt. To trigger this manually, issue:

```
% fc-cache
```

Plenty of free, high quality fonts of both types are available in the ports tree which can be readily used with X Window System. This chapter provides a brief overview of both, as well as configuring the X FreeType interface.

For more information about how to install and configure fonts on FreeBSD, please read the article Fonts and FreeBSD.

5.6.1. TrueType® Fonts

X.org has built in support for rendering TrueType® fonts. There are two different modules that can enable this functionality. The freetype module is used in this example because it is more consistent with the other font rendering back-ends. To enable the freetype module just add the following line to the `"Module"` section of /usr/local/etc/X11/xorg.conf.d/90-fonts.conf.

```
Load  "freetype"
```

Now make a directory for the TrueType® fonts (for example, /usr/local/share/fonts/TrueType) and copy all of the TrueType® fonts into this directory. Keep in mind that TrueType® fonts cannot be directly taken from an Apple® Mac®; they must be in UNIX®/MS-DOS®/Windows® format for use by X.org. Once the files have been copied into this directory, use mkfontscale to create a fonts.dir, so that the X font renderer knows that these new files have been installed. `mkfontscale` can be installed as a package:

```
# pkg install mkfontscale
```

Then create an index of X font files in a directory:

```
# cd /usr/local/share/fonts/TrueType
# mkfontscale
```

Now add the TrueType® directory to the font path. This is just the same as described in Type1 Fonts:

```
% xset fp+ /usr/local/share/fonts/TrueType
% xset fp rehash
```

or add a `FontPath` line to xorg.conf.

Now Gimp, LibreOffice, and all of the other X applications should now recognize the installed TrueType® fonts. Extremely small fonts (as with text in a high resolution display on a web page) and extremely large fonts (within LibreOffice) will look much better now.

5.6.2. Type1 Fonts

The URW font collection (x11-fonts/urwfonts) includes high quality versions of standard type1 fonts (Times Roman™, Helvetica™, Palatino™ and others). The Freefonts collection (x11-fonts/freefonts) includes many more fonts, but most of them are intended for use in graphics software such as the Gimp, and are not complete enough to serve as screen fonts.

To install the above Type1 font collections from binary packages, run the following commands:

```
# pkg install urwfonts
```

And likewise with the freefont or other collections. To have a manually configured X server detect these fonts, add an appropriate line to the X server configuration file (/usr/local/etc/X11/xorg.conf.d/90-fonts.conf), which reads:

```
Section "Files"
  FontPath "/usr/local/share/fonts/urwfonts/"
EndSection
```

Alternatively, at the command line in the X session run:

```
% xset fp+ /usr/local/share/fonts/urwfonts
% xset fp rehash
```

This will work but will be lost when the X session is closed, unless it is added to the startup file (~/.xinitrc for a normal `startx` session, or ~/.xsession when logging in through a graphical login manager like XDM). A third way is to use the new /usr/local/etc/fonts/local.conf as demonstrated in Anti-Aliased Fonts.

5.6.3. Anti-Aliased Fonts

All fonts in X.org that are found in /usr/local/share/fonts/ and ~/.fonts/ are automatically made available for anti-aliasing to Xft-aware applications. Most recent applications are Xft-aware, including KDE, GNOME, and Firefox.

To control which fonts are anti-aliased, or to configure anti-aliasing properties, create (or edit, if it already exists) the file /usr/local/etc/fonts/local.conf. Several advanced features of the Xft font system can be tuned using this file; this section describes only some simple possibilities. For more details, please see fonts-conf(5).

This file must be in XML format. Pay careful attention to case, and make sure all tags are properly closed. The file begins with the usual XML header followed by a DOCTYPE definition, and then the `<fontconfig>` tag:

```
<?xml version="1.0"?>
      <!DOCTYPE fontconfig SYSTEM "fonts.dtd">
      <fontconfig>
```

As previously stated, all fonts in /usr/local/share/fonts/ as well as ~/.fonts/ are already made available to Xft-aware applications. To add another directory outside of these two directory trees, add a line like this to /usr/local/etc/fonts/local.conf:

```
<dir>/path/to/my/fonts</dir>
```

After adding new fonts, and especially new font directories, rebuild the font caches:

```
# fc-cache -f
```

Anti-aliasing makes borders slightly fuzzy, which makes very small text more readable and removes "staircases" from large text, but can cause eyestrain if applied to normal text. To exclude font sizes smaller than 14 point from anti-aliasing, include these lines:

```
    <match target="font">
        <test name="size" compare="less">
        <double>14</double>
        </test>
        <edit name="antialias" mode="assign">
        <bool>false</bool>
        </edit>
    </match>
    <match target="font">
        <test name="pixelsize" compare="less" qual="any">
        <double>14</double>
        </test>
        <edit mode="assign" name="antialias">
        <bool>false</bool>
        </edit>
    </match>
```

Spacing for some monospaced fonts might also be inappropriate with anti-aliasing. This seems to be an issue with KDE, in particular. One possible fix is to force the spacing for such fonts to be 100. Add these lines:

```
    <match target="pattern" name="family">
       <test qual="any" name="family">
           <string>fixed</string>
       </test>
       <edit name="family" mode="assign">
           <string>mono</string>
       </edit>
    </match>
    <match target="pattern" name="family">
        <test qual="any" name="family">
        <string>console</string>
        </test>
        <edit name="family" mode="assign">
        <string>mono</string>
        </edit>
    </match>
```

(this aliases the other common names for fixed fonts as `"mono"`), and then add:

```
    <match target="pattern" name="family">
         <test qual="any" name="family">
         <string>mono</string>
         </test>
         <edit name="spacing" mode="assign">
         <int>100</int>
         </edit>
     </match>
```

Certain fonts, such as Helvetica, may have a problem when anti-aliased. Usually this manifests itself as a font that seems cut in half vertically. At worst, it may cause applications to crash. To avoid this, consider adding the following to local.conf:

```
    <match target="pattern" name="family">
         <test qual="any" name="family">
         <string>Helvetica</string>
         </test>
         <edit name="family" mode="assign">
         <string>sans-serif</string>
         </edit>
     </match>
```

After editing local.conf, make certain to end the file with the `</fontconfig>` tag. Not doing this will cause changes to be ignored.

Users can add personalized settings by creating their own ~/.config/fontconfig/fonts.conf. This file uses the same `XML` format described above.

One last point: with an LCD screen, sub-pixel sampling may be desired. This basically treats the (horizontally separated) red, green and blue components separately to improve the horizontal resolution; the results can be dramatic. To enable this, add the line somewhere in local.conf:

```
<match target="font">
    <test qual="all" name="rgba">
    <const>unknown</const>
    </test>
    <edit name="rgba" mode="assign">
    <const>rgb</const>
    </edit>
</match>
```

ℹ Depending on the sort of display, `rgb` may need to be changed to `bgr`, `vrgb` or `vbgr`: experiment and see which works best.

Chapter 6. Wayland on FreeBSD

6.1. Synopsis

An installation of FreeBSD using bsdinstall does not automatically install a graphical user interface. This chapter describes how to select, install, and configure a Wayland compositor, which provides a graphical environment.

Before reading this chapter:

- Know how to install additional third-party software.
- Know how to identify and configure drivers for the graphics hardware.

Read this chapter to learn:

- How to configure FreeBSD to host a Wayland graphical environment.
- How to install and configure a Wayland compositor.
- How to run programs designed for the older X Window System.
- How to configure remote desktop access to a Wayland graphical environment.

6.2. Wayland Overview

Wayland is a communication protocol that can replace a display server such as Xorg. It differs from Xorg in several important ways. First, Wayland is only a protocol that acts as an intermediary between clients using a mechanism which removes the dependency on an X server. Xorg includes both the X11 protocol, used to run remote displays, and the X server, used to accept connections and display windows. Under Wayland, the compositor or window manager provides the display server instead of a traditional X server.

Since Wayland is not an X server, traditional X screen connections will need to utilize other methods such as VNC or RDP for remote desktop management. Second, Wayland can manage composite communications between clients and a compositor as a separate entity which does not need to support the X protocols.

Wayland is relatively new, and not all software has been updated to run natively without `Xwayland` support. Because Wayland does not provide the X server, and expects compositors to provide that support, X11 window managers that do not yet support Wayland will require that `Xwayland` is not started with the `-rootless` parameter. The `-rootless` parameter, when removed, will restore X11 window manager support.

> The current NVIDIA® driver should work with most wlroots compositors, but it may be a little unstable and not support all features at this time. Volunteers to help work on the NVIDIA® DRM are requested.

Currently, a lot of software will function with minimal issues on Wayland, including Firefox. And a few desktops are also available, such as the Compiz Fusion replacement, known as Wayfire, and the i3 window manager replacement, Sway.

As of May, 2021, plasma5-kwin does support Wayland on FreeBSD. To use Plasma under Wayland, use the `startplasma-wayland` parameter to `ck-launch-session` and tie in dbus with: `dbus-launch --exit-with-x11 ck-launch-session startplasma-wayland` to get it working.

For compositors, a kernel supporting the evdev(4) driver must exist to utilize the keybinding functionality. This is built into the GENERIC kernel by default; however, if it has been customized and evdev(4) support was stripped out, the evdev(4) module will need to be loaded. In addition, users of `Wayland` will need to be members of the `video` group. To quickly make this change, use the `pw` command:

```
pw groupmod video -m user
```

Installing Wayland is simple; there is not a great deal of configuration for the protocol itself. Most of the composition will depend on the chosen compositor. By installing `seatd` now, a step is skipped as part of the compositor installation and configuration as `seatd` is needed to provide non-root access to certain devices.

All of the compositors described here should work with graphics/drm-kmod open source drivers; however, the NVIDIA® graphics cards may have issues when using the proprietary drivers. Begin by installing the following packages:

```
# pkg install wayland seatd
```

Once the protocol and supporting packages have been installed, a compositor must create the user interface. Several compositors will be covered in the following sections. All compositors using Wayland will need a runtime directory defined in the environment. Since FreeBSD 14.1, this is created and defined automatically. For previous versions, this is achieved with the following command in the bourne shell:

```
% export XDG_RUNTIME_DIR=/var/run/user/`id -u`
```

It is important to note that most compositors will search the XDG_RUNTIME_DIR directory for the configuration files. In the examples included here, a parameter will be used to specify a configuration file in ~/.config to keep temporary files and configuration files separate. It is recommended that an alias be configured for each compositor to load the designated configuration file.

It has been reported that ZFS users may experience issues with some Wayland clients because they need access to `posix_fallocate()` in the runtime directory. While the author could not reproduce this issue on their ZFS system, a recommended workaround is not to use ZFS for the runtime directory and instead use `tmpfs` for the /var/run directory. In this case, the `tmpfs` file system is used for

> /var/run and mounted through the command `mount -t tmpfs tmpfs /var/run` command and then make this change persist across reboots through /etc/fstab. The XDG_RUNTIME_DIR environment variable could be configured to use /var/run/user/$UID and avoid potential pitfalls with ZFS. Consider that scenario when reviewing the configuration examples in the following sections.

The seatd daemon helps manage access to shared system devices for non-root users in compositors; this includes graphics cards. For traditional X11 managers, `seatd` is not needed, such as both Plasma and GNOME, but for the Wayland compositors discussed here, it will need enabled on the system and be running before starting a compositor environment. To enable and start the `seatd` daemon now, and on system initialization:

```
# sysrc seatd_enable="YES"
# service seatd start
```

Afterward, a compositor, which is similar to an X11 desktop, will need to be installed for the GUI environment. Three are discussed here, including basic configuration options, setting up a screen lock, and recommendations for more information.

6.3. The Wayfire Compositor

Wayfire is a compositor that aims to be lightweight and customizable. Several features are available, and it brings back several elements from the previously released Compiz Fusion desktop. All of the parts look beautiful on modern hardware. To get Wayfire up and running, begin by installing the required packages:

```
# pkg install wayfire wf-shell alacritty swaylock-effects swayidle wlogout kanshi mako
wlsunset
```

The `alacritty` package provides a terminal emulator. Still, it is not completely required as other terminal emulators such as `kitty`, and XFCE-4 `Terminal` have been tested and verified to work under the Wayfire compositor. Wayfire configuration is relatively simple; it uses a file that should be reviewed for any customizations. To begin, copy the example file over to the runtime environment configuration directory and then edit the file:

```
% mkdir ~/.config/wayfire
% cp /usr/local/share/examples/wayfire/wayfire.ini ~/.config/wayfire
```

The defaults for most users should be fine. Within the configuration file, items like the famous `cube` are pre-configured, and there are instructions to help with the available settings. A few primary settings of note include:

```
[output]
mode = 1920x1080@60000
position = 0,0
```

```
transform = normal
scale = 1.000000
```

In this example, from the configuration file, the screen's output should be the listed mode at the listed hertz. For example, the mode should be set to `widthxheight@refresh_rate`. The position places the output at a specific pixel location specified. The default should be fine for most users. Finally, transform sets a background transform, and scale will scale the output to the specified scale factor. The defaults for these options are generally acceptable; for more information, see the documentation.

As mentioned, Wayland is new, and not all applications work with the protocol yet. At this time, `sddm` does not appear to support starting and managing compositors in Wayland. The `swaylock` utility has been used instead in these examples. The configuration file contains options to run `swayidle` and `swaylock` for idle and locking of the screen.

This option to define the action to take when the system is idle is listed as:

```
idle = swaylock
```

And the lock timeout is configured using the following lines:

```
[idle]
toggle = <super> KEY_Z
screensaver_timeout = 300
dpms_timeout = 600
```

The first option will lock the screen after 300 seconds, and after another 300, the screen will shut off through the `dpms_timeout` option.

One final thing to note is the <super> key. Most of the configuration mentions this key, and it is the traditional `Windows` key on the keyboard. Most keyboards have this super key available; however, it should be remapped within this configuration file if it is not available. For example, to lock the screen, press and hold the super key, the `shift` key, and press the `escape` key. Unless the mappings have changed, this will execute the swaylock application. The default configuration for `swaylock` will show a grey screen; however, the application is highly customizable and well documented. In addition, since the swaylock-effects is the version that was installed, there are several options available such as the blur effect, which can be seen using the following command:

```
% swaylock --effect-blur 7x5
```

There is also the `--clock` parameter which will display a clock with the date and time on the lock screen. When x11/swaylock-effects was installed, a default pam.d configuration was included. It provides the default options that should be fine for most users. More advanced options are available; see the PAM documentation for more information.

At this point, it is time to test Wayfire and see if it can start up on the system. Just type the following command:

```
% wayfire -c ~/.config/wayfire/wayfire.ini
```

The compositor should now start and display a background image along with a menu bar at the top of the screen. Wayfire will attempt to list installed compatible applications for the desktop and present them in this drop-down menu; for example, if the XFCE-4 file manager is installed, it will show up in this drop-down menu. If a specific application is compatible and valuable enough for a keyboard shortcut, it may be mapped to a keyboard sequence using the wayfire.ini configuration file. Wayfire also has a configuration tool named Wayfire Config Manager. It is located in the drop-down menu bar but may also be started through a terminal by issuing the following command:

```
% wcm
```

Various Wayfire configuration options, including the composite special effects, maybe enabled, disabled, or configured through this application. In addition, for a more user-friendly experience, a background manager, panel, and docking application may be enabled in the configuration file:

```
panel = wf-panel
dock = wf-dock
background = wf-background
```

> Changes made through `wcm` will overwrite custom changes in the wayfire.ini configuration file. The wayfire.ini file is highly recommended to be backed up so any essential changes may be restored.

Finally, the default launcher listed in the wayfire.ini is x11/wf-shell which may be replaced with other panels if desired by the user.

6.4. The Hikari Compositor

The Hikari compositor uses several concepts centered around productivity, such as sheets, workspaces, and more. In that way, it resembles a tiling window manager. Breaking this down, the compositor starts with a single workspace, which is similar to virtual desktops. Hikari uses a single workspace or virtual desktop for user interaction. The workspace is made up of several views, which are the working windows in the compositor grouped as either sheets or groups. Both sheets and groups are made up of a collection of views; again, the windows that are grouped together. When switching between sheets or groups, the active sheet or group will become known collectively as the workspace. The manual page will break this down into more information on the functions of each but for this document, just consider a single workspace utilizing a single sheet. Hikari installation will comprise of a single package, x11-wm/hikari, and a terminal emulator `alacritty`:

```
# pkg install hikari alacritty
```

Other shells, such as `kitty` or the Plasma `Terminal`, will function under Wayland. Users should experiment with their favorite terminal editor to validate compatibility.

Hikari uses a configuration file, hikari.conf, which could either be placed in the XDG_RUNTIME_DIR or specified on startup using the `-c` parameter. An autostart configuration file is not required but may make the migration to this compositor a little easier. Beginning the configuration is to create the Hikari configuration directory and copy over the configuration file for editing:

```
% mkdir ~/.config/hikari
% cp /usr/local/etc/hikari/hikari.conf ~/.config/hikari
```

The configuration is broken out into various stanzas such as ui, outputs, layouts, and more. For most users, the defaults will function fine; however, some important changes should be made. For example, the $TERMINAL variable is normally not set within the user's environment. Changing this variable or altering the hikari.conf file to read:

```
terminal = "/usr/local/bin/alacritty"
```

Will launch the `alacritty` terminal using the bound key press. While going through the configuration file, it should be noted that the capital letters are used to map keys out for the user. For example, the `L` key for starting the terminal `L` + `Return` is actually the previously discussed super key or Windows logo key. Therefore, holding the `L/super/Windows` key and pressing `Enter` will open the specified terminal emulator with the default configuration. Mapping other keys to applications require an action definition to be created. For this, the action item should be listed in the actions stanza, for example:

```
actions {
  terminal = "/usr/local/bin/alacritty"
  browser = "/usr/local/bin/firefox"
}
```

Then an action may be mapped under the keyboard stanza, which is defined within the bindings stanza:

```
bindings {
  keyboard {
SNIP
    "L+Return" = action-terminal
    "L+b" = action-browser
SNIP
```

After Hikari is restarted, holding the Windows logo button and pressing the `b` key on the keyboard will start the web browser. The compositor does not have a menu bar, and it is recommended the user set up, at minimal, a terminal emulator before migration. The manual page contains a great deal of documentation it should be read before performing a full migration. Another positive aspect about Hikari is that, while migrating to the compositor, Hikari can be started in the Plasma and GNOME desktop environments, allowing for a test-drive before completely migrating.

Locking the screen in Hikari is easy because a default pam.d configuration file and unlock utility are bundled with the package. The key binding for locking the screen is `L` (Windows logo key)+ `Shift` + `Backspace`. It should be noted that all views not marked public will be hidden. These views will never accept input when locked but beware of sensitive information being visible. For some users, it may be easier to migrate to a different screen locking utility such as swaylock-effects, discussed in this section. To start Hikari, use the following command:

```
% hikari -c ~/.config/hikari/hikari.conf
```

6.5. The Sway Compositor

The Sway compositor is a tiling compositor that attempts to replace the i3 windows manager. It should work with the user's current i3 configuration; however, new features may require some additional setup.

Before starting the Sway installation, ensure that a graphics card (GPU) is installed and configured correctly. Refer to the section drivers for the graphics hardware for guidance. This step is essential for the Sway compositor to function properly.

In the forthcoming examples, a fresh installation without migrating any i3 configuration will be assumed. To install Sway and valuable components, issue the following command as the root user:

```
# pkg install sway swayidle swaylock-effects alacritty dmenu-wayland dmenu
```

For a basic configuration file, issue the following commands and then edit the configuration file after it is copied:

```
% mkdir ~/.config/sway
% cp /usr/local/etc/sway/config ~/.config/sway
```

The base configuration file has many defaults, which will be fine for most users. Several important changes should be made like the following:

```
# Logo key. Use Mod1 for Alt.
input * xkb_rules evdev
set $mod Mod4
# The preferred terminal emulator
set $term alacritty
set $lock swaylock -f -c 000000
output "My Workstation" mode 1366x768@60Hz position 1366 0
output * bg ~/wallpapers/mywallpaper.png stretch
### Idle configuration
exec swayidle -w \
          timeout 300 'swaylock -f -c 000000' \
          timeout 600 'swaymsg "output * dpms off"' resume 'swaymsg "output * dpms
on"' \
          before-sleep 'swaylock -f -c 000000'
```

In the previous example, the `xkb` rules for evdev(4) events are loaded, and the $mod key is set to the Windows logo key for the key bindings. Next, the terminal emulator was set to be `alacritty`, and a screen lock command was defined; more on this later. The output keyword, the mode, the position, a background wallpaper, and Sway is also told to stretch this wallpaper to fill out the screen. Finally, `swayidle` is set to daemonize and lock the screen after a timeout of 300 seconds, placing the screen or monitor into sleep mode after 600 seconds. The locked background color of 000000, which is black, is also defined here. Using swaylock-effects, a clock may also be displayed with the `--clock` parameter. See the manual page for more options. The sway-output(5) manual page should also be reviewed; it includes a great deal of information on customing the output options available.

While in Sway, to bring up a menu of applications, hold the Windows logo key (mod) and press the `d` key. The menu may be navigated using the arrow keys on the keyboard. There is also a method to manipulate the layout of the bar and add a tray; read the sway-bar(5) manual page for more information. The default configuration adds a date and time to the upper right-hand corner. See the `Bar` stanza in the configuration file for an example. By default, the configuration does not include locking the screen outside of the example above, enabling a lockout timer. Creating a lock key binding requires the following line to the `Key bindings` section:

```
# Lock the screen manually
bindsym $mod+Shift+Return exec $lock
```

Now the screen may be locked using the combination of holding the Windows logo key, pressing and holding shift, and finally pressing return. When Sway is installed, whether from a package or the FreeBSD Ports Collection, a default file for pam.d was installed. The default configuration should be acceptable for most users, but more advanced options are available. Read through the PAM documentation for more information.

Finally, to exit Sway and return to the shell, hold the Windows logo key, the shift key, and press the `e` key. A prompt will be displayed with an option to exit Sway. During migration, Sway can be started through a terminal emulator on an X11 desktop such as Plasma. This makes testing different changes and key bindings a little easier prior to fully migrating to this compositor. To start Sway, issue the following command:

```
% sway -c ~/.config/sway/config
```

6.6. Using Xwayland

When installing Wayland, the `Xwayland` binary should have been installed unless Wayland was built without X11 support. If the /usr/local/bin/Xwayland file does not exist, install it using the following command:

```
# pkg install xwayland
```

The development version of Xwayland is recommended and was most likely installed with the Wayland package. Each compositor has a method of enabling or disabling this feature.

Once `Xwayland` has been installed, configure it within the chosen compositor. For Wayfire, the following line is required in the wayfire.ini file:

```
xwayland = true
```

For the Sway compositor, `Xwayland` should be enabled by default. Even so, it is recommended to manually add a configuration line in the ~/.config/sway/config like the following:

```
xwayland enable
```

Finally, for Hikari, no changes are needed. Support for `Xwayland` is built in by default. To disable that support, rebuild the package from the ports collection and disable Xwayland support at that time.

After these changes are made, start the compositor at the command line and execute a terminal from the key bindings. Within this terminal, issue the `env` command and search for the `DISPLAY` variables. If the compositor was able to properly start the Xwayland X server, these environment variables should look similar to the following:

```
% env | grep DISPLAY
```

```
WAYLAND_DISPLAY=wayland-1
DISPLAY=:0
```

In this output, there is a default Wayland display and a display set for the Xwayland server. Another method to verify that `Xwayland` is functioning properly is to use install and test the small package:[x11/eyes] and check the output. If the `xeyes` application starts and the eyes follow the mouse pointer, Xwayland is functioning properly. If an error such as the following is displayed, something happened during the `Xwayland` initialization and it may need reinstalled:

```
Error: Cannot open display wayland-0
```

A security feature of Wayland is that, without running an X server, there is not another network listener. Once `Xwayland` is enabled, this security feature is no

> longer applicable to the system.

For some compositors, such as Wayfire, `Xwayland` may not start properly. As such, `env` will show the following information for the `DISPLAY` environment variables:

```
% env | grep DISPLAY
```

```
DISPLAY=wayland-1
WAYLAND_DISPLAY=wayland-1
```

Even though `Xwayfire` was installed and configured, X11 applications will not start giving a display issue. To work around this, verify that there is already an instance of `Xwayland` using a UNIX socket through these two methods. First, check the output from `sockstat` and search for X11-unix:

```
% sockstat | grep x11
```

There should be something similar to the following information:

```
trhodes  Xwayland   2734  8  stream /tmp/.X11-unix/X0
trhodes  Xwayland   2734  9  stream /tmp/.X11-unix/X0
trhodes  Xwayland   2734  10 stream /tmp/.X11-unix/X0
trhodes  Xwayland   2734  27 stream /tmp/.X11-unix/X0_
trhodes  Xwayland   2734  28 stream /tmp/.X11-unix/X0
```

This suggests the existence of an X11 socket. This can be further verified by attempting to execute `Xwayland` manually within a terminal emulator running under the compositor:

```
% Xwayland
```

If an X11 socket is already available, the following error should be presented to the user:

```
(EE)
Fatal server error:
(EE) Server is already active for display 0
    If this server is no longer running, remove /tmp/.X0-lock
    and start again.
(EE)
```

Since there is an active X display available using display zero, the environment variable was just set improperly, to fix this, change the `DISPLAY` environment variable to `:0` and attempt to execute the application again. The following example uses mail/claws-mail as the application which needs the `Xwayland` service:

```
export DISPLAY=:0
```

After this change, the mail/claws-mail application should now start using `Xwayland` and function as expected.

6.7. Remote Desktop Using VNC

Earlier in this document it was noted that Wayland does not provide the same X server style access as Xorg provides. Instead, users are free to pick and choose a remote desktop protocol such as RDP or VNC. The FreeBSD Ports collection includes the `wayvnc`, which will support wlroots based compositors such as the ones discussed here. This application may be installed using:

```
# pkg install wayvnc
```

Unlike some other packages, `wayvnc` does not come with a configuration file. Thankfully, the manual page documents the important options and they may be extrapolated into a simple configuration file:

```
address=0.0.0.0
enable_auth=true
username=username
password=password
private_key_file=/path/to/key.pem
certificate_file=/path/to/cert.pem
```

The key files will need to be generated, and it is highly recommended they be used for increased security of the connection. When invoked, wayvnc will search for the configuration file in ~/.config/wayvnc/config. This could be overwritten using the `-C configuration_file` option when starting the server. Thus, to start the `wayvnc` server, issue the following command:

```
% wayvnc -C ~/.config/wayvnc/config
```

At the time of this writing, there is no rc.d script to start `wayvnc` on system initialization. If that functionality is desired, a local startup file will need to be created. This is probably a feature request for the port maintainer.

6.8. Wayland Login Manager

While several login managers exist and are slowly migrating to Wayland, one option is the x11/ly text user interface (TUI) manager. Needing minimal configuration, `ly` will start Sway, Wayfire, and others by presenting a login window on system initialization. To install `ly`, issue the following command:

```
# pkg install ly
```

There will be some configuration hints presented, the import steps are to add the following lines to /etc/gettytab:

```
Ly:\
  :lo=/usr/local/bin/ly:\
  :al=root:
```

And then modify the ttyv1 line in /etc/ttys to match the following line:

```
ttyv1 "/usr/libexec/getty Ly" xterm onifexists secure
```

After a system reboot, a login should appear. To configure specific settings, such as language and edit /usr/local/etc/ly/config.ini. At minimal, this file should have the designated tty that was previously specified in /etc/ttys.

> If setting ttyv0 up as the login terminal, it may be required to press the `alt` and `F1` keys to properly see the login window.

When the login window appears, using the left and right arrows will swap through different, supported, window managers.

6.9. Useful Utilities

One useful Wayland utility which all compositors can make use of is the waybar. While Wayfire does come with a launch menu, an easy-to-use and fast taskbar is a good accessory for any compositor or desktop manager. A Wayland compatible taskbar that is fast and easy to configure is waybar. To install the package and a supporting audio control utility, issue the following command:

```
# pkg install pavucontrol waybar
```

To create the configuration directory and copy over a default configuration file, execute the following commands:

```
% mkdir ~/.config/waybar
% cp /usr/local/etc/xdg/waybar/config ~/.config/waybar
```

The `lavalauncher` utility provides a launch bar for various applications. There is no example configuration file provided with the package, so the following actions must be taken:

```
mkdir ~/.config/lavalauncher
```

An example configuration file that only includes Firefox, and is placed on the right, is below:

```
global-settings {
    watch-config-file = true;
}

bar {
    output            = eDP-1;
    position          = bottom;
    background-colour = "#202020";

    # Condition for the default configuration set.
    condition-resolution = wider-than-high;

    config {
        position = right;
    }

    button {
        image-path          =
/usr/local/lib/firefox/browser/chrome/icons/default/default48.png;
        command[mouse-left] =     /usr/local/bin/firefox;
    }
    button {
      image-path           =   /usr/local/share/pixmaps/thunderbird.png;
      command[mouse-left]  =   /usr/local/bin/thunderbird;
}
```

Chapter 7. Network

7.1. Synopsis

This chapter delves into the topic of network configuration and performance, showcasing the robust networking capabilities of the FreeBSD operating system. Whether working with wired or wireless networks, this chapter provides a comprehensive guide to configuring and optimizing network connectivity in FreeBSD.

Before diving into the details, it is beneficial for readers to have a basic understanding of networking concepts such as protocols, network interfaces, and addressing.

This chapter covers:

- The ability to configure wired networks in FreeBSD, including network interface setup, addressing, and customization options.
- The skills to configure wireless networks in FreeBSD, encompassing wireless network interface setup, security protocols, and troubleshooting techniques.
- FreeBSD's networking capabilities and its reputation for excellent network performance.
- An understanding of various network services and protocols supported by FreeBSD, with configuration instructions for DNS, DHCP and more.

More information about how to make advanced network configurations in Advanced Networking.

7.2. Setting up the Network

Setting up a wired or wireless connection is a common task for a FreeBSD user. This section will show how to identify the wired and wireless network adapters and how to configure them.

Before starting with the configuration it is necessary to know the following network data:

- If the network has DHCP
- If the network does not have DHCP, the static IP to be used
- The netmask
- The IP address of the default gateway

> The network connection may have been configured at installation time by bsdinstall(8).

7.2.1. Identify Network Adapters

FreeBSD supports a wide variety of network adapters for both wired and wireless networks. Check the Hardware Compatibility List for the used FreeBSD release to see if the network adapter is supported.

To get the network adapters used by our system execute the following command:

```
% pciconf -lv | grep -A1 -B3 network
```

The output should be similar to the following:

```
em0@pci0:0:25:0:        class=0x020000 rev=0x03 hdr=0x00 vendor=0x8086 device=0x10f5
subvendor=0x17aa subdevice=0x20ee
    vendor     = 'Intel Corporation' ①
    device     = '82567LM Gigabit Network Connection' ②
    class      = network
    subclass   = ethernet
--
iwm0@pci0:3:0:0:        class=0x028000 rev=0x00 hdr=0x00 vendor=0x8086 device=0x4237
subvendor=0x8086 subdevice=0x1211
    vendor     = 'Intel Corporation' ①
    device     = 'PRO/Wireless 5100 AGN [Shiloh] Network Connection' ②
    class      = network
```

The text before the '@' symbol is the name of the driver controlling the device. In this case these are em(4) and iwn(4).

① Shows the name of the vendor

② Shows the name of the device

It is only necessary to load the network interface card module if FreeBSD has not detected it correctly.

For example, to load the alc(4) module, execute the following command:

```
# kldload if_alc
```

Alternatively, to load the driver as a module at boot time, place the following line in /boot/loader.conf:

```
if_alc_load="YES"
```

7.3. Wired Networks

Once the right driver is loaded the network adapter needs to be configured. FreeBSD uses the driver name followed by a unit number to name the network interface adapter. The unit number represents the order in which the adapter is detected at boot time, or is later discovered.

For example, `em0` is the first network interface card (NIC) on the system using the em(4) driver.

To display the network interface configuration, enter the following command:

```
% ifconfig
```

The output should be similar to the following:

```
em0: flags=8863<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500

options=481249b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,LRO,WOL_MAGIC,VLAN_H
WFILTER,NOMAP>
        ether 00:1f:16:0f:27:5a
        inet6 fe80::21f:16ff:fe0f:275a%em0 prefixlen 64 scopeid 0x1
        inet 192.168.1.19 netmask 0xffffff00 broadcast 192.168.1.255
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active
        nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> metric 0 mtu 16384
        options=680003<RXCSUM,TXCSUM,LINKSTATE,RXCSUM_IPV6,TXCSUM_IPV6>
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x2
        inet 127.0.0.1 netmask 0xff000000
        groups: lo
        nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
```

In this example, the following devices were displayed:

- `em0`: The Ethernet interface.
- `lo0`: The loop interface is a software loopback mechanism which may be used for performance analysis, software testing, and/or local communication. More information in lo(4).

The example shows that `em0` is up and running.

The key indicators are:

1. `UP` means that the interface is configured and ready.
2. The interface has an IPv4 Internet (`inet`) address, `192.168.1.19`.
3. The interface has an IPv6 Internet (`inet6`) address, `fe80::21f:16ff:fe0f:275a%em0`.
4. It has a valid subnet mask (`netmask`), where `0xffffff00` is the same as `255.255.255.0`.
5. It has a valid broadcast address, `192.168.1.255`.
6. The MAC address of the interface (`ether`) is `00:1f:16:0f:27:5a`.
7. The physical media selection is on autoselection mode (`media: Ethernet autoselect (1000baseT <full-duplex>)`).
8. The status of the link (`status`) is `active`, indicating that the carrier signal is detected. For `em0`, the `status: no carrier` status is normal when an Ethernet cable is not plugged into the interface.

If the ifconfig(8) output had shown something similar to the next output it would indicate the interface has not been configured:

```
em0: flags=8822<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500

options=481249b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,LRO,WOL_MAGIC,VLAN_H
WFILTER,NOMAP>
        ether 00:1f:16:0f:27:5a
        media: Ethernet autoselect
        status: no carrier
        nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

7.3.1. Configuring Static IPv4 Address

This section provides a guide to configuring a static IPv4 address on a FreeBSD system.

The network interface card configuration can be performed from the command line with ifconfig(8) but will not persist after a reboot unless the configuration is also added to /etc/rc.conf.

If the network was configured during installation by bsdinstall(8), some entries for the network interface card (NICs) may be already present. Double check /etc/rc.conf before executing sysrc(8).

The IP address can be set executing the following command:

```
# ifconfig em0 inet 192.168.1.150/24
```

To make the change persist across reboots execute the following command:

```
# sysrc ifconfig_em0="inet 192.168.1.150 netmask 255.255.255.0"
```

Add the default router executing the following command:

```
# sysrc defaultrouter="192.168.1.1"
```

Add the DNS records to /etc/resolv.conf:

```
nameserver 8.8.8.8
nameserver 8.8.4.4
```

Then restart `netif` and `routing` executing the following command:

```
# service netif restart && service routing restart
```

The connection can be tested using ping(8):

```
% ping -c2 www.FreeBSD.org
```

The output should be similar to the following:

```
PING web.geo.FreeBSD.org (147.28.184.45): 56 data bytes
64 bytes from 147.28.184.45: icmp_seq=0 ttl=51 time=55.173 ms
64 bytes from 147.28.184.45: icmp_seq=1 ttl=51 time=53.093 ms

--- web.geo.FreeBSD.org ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 53.093/54.133/55.173/1.040 ms
```

7.3.2. Configuring Dynamic IPv4 Address

If the network has a DHCP server, it is very easy to configure the network interface to use DHCP. FreeBSD uses dhclient(8) as the DHCP client. dhclient(8) will automatically provide the IP, the netmask and the default router.

To make the interface work with DHCP execute the following command:

```
# sysrc ifconfig_em0="DHCP"
```

dhclient(8) can be used manually by running the following command:

```
# dhclient em0
```

The output should be similar to the following:

```
DHCPREQUEST on em0 to 255.255.255.255 port 67
DHCPACK from 192.168.1.1
unknown dhcp option value 0x7d
bound to 192.168.1.19 -- renewal in 43200 seconds.
```

In this way it can be verified that the address assignment using DHCP works correctly.

> dhclient(8) client can be started in background. This can cause trouble with applications depending on a working network, but it will provide a faster startup in many cases.
>
> To execute dhclient(8) in background execute the following command:

```
# sysrc background_dhclient="YES"
```

Then restart `netif` executing the following command:

```
# service netif restart
```

The connection can be tested using ping(8):

```
% ping -c2 www.FreeBSD.org
```

The output should be similar to the following:

```
PING web.geo.FreeBSD.org (147.28.184.45): 56 data bytes
64 bytes from 147.28.184.45: icmp_seq=0 ttl=51 time=55.173 ms
64 bytes from 147.28.184.45: icmp_seq=1 ttl=51 time=53.093 ms

--- web.geo.FreeBSD.org ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 53.093/54.133/55.173/1.040 ms
```

7.3.3. IPv6

IPv6 is the new version of the well-known IP protocol, also known as IPv4.

IPv6 provides several advantages over IPv4 as well as many new features:

- Its 128-bit address space allows for 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses. This addresses the IPv4 address shortage and eventual IPv4 address exhaustion.
- Routers only store network aggregation addresses in their routing tables, thus reducing the average space of a routing table to 8192 entries. This addresses the scalability issues associated with IPv4, which required every allocated block of IPv4 addresses to be exchanged between Internet routers, causing their routing tables to become too large to allow efficient routing.
- Address autoconfiguration (RFC4862).
- Mandatory multicast addresses.
- Built-in IPsec (IP security).
- Simplified header structure.
- Support for mobile IP.
- IPv6-to-IPv4 transition mechanisms.

FreeBSD includes the KAME project IPv6 reference implementation and comes with everything needed to use IPv6.

This section focuses on getting IPv6 configured and running.

There are three different types of IPv6 addresses:

Unicast

A packet sent to a unicast address arrives at the interface belonging to the address.

Anycast

These addresses are syntactically indistinguishable from unicast addresses but they address a group of interfaces. The packet destined for an anycast address will arrive at the nearest interface.

Multicast

These addresses identify a group of interfaces. A packet destined for a multicast address will arrive at all interfaces belonging to the multicast group. The IPv4 broadcast address, usually `xxx.xxx.xxx.255`, is expressed by multicast addresses in IPv6.

When reading an IPv6 address, the canonical form is represented as `x:x:x:x:x:x:x:x`, where each `x` represents a 16 bit hex value. An example is `FEBC:A574:382B:23C1:AA49:4592:4EFE:9982`.

Often, an address will have long substrings of all zeros. A `::` (double colon) can be used to replace one substring per address. Also, up to three leading `0`s per hex value can be omitted. For example, `fe80::1` corresponds to the canonical form `fe80:0000:0000:0000:0000:0000:0000:0001`.

A third form is to write the last 32 bits using the well known IPv4 notation. For example, `2002::10.0.0.1` corresponds to the hexadecimal canonical representation `2002:0000:0000:0000:0000:0000:0a00:0001`, which in turn is equivalent to `2002::a00:1`.

To view a FreeBSD system's IPv6 address execute the following command:

```
# ifconfig
```

The output should be similar to the following:

```
em0: flags=8863<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500

options=481249b<RXCSUM,TXCSUM,VLAN_MTU,VLAN_HWTAGGING,VLAN_HWCSUM,LRO,WOL_MAGIC,VLAN_H
WFILTER,NOMAP>
        ether 00:1f:16:0f:27:5a
        inet 192.168.1.150 netmask 0xffffff00 broadcast 192.168.1.255
        inet6 fe80::21f:16ff:fe0f:275a%em0 prefixlen 64 scopeid 0x1
        media: Ethernet autoselect (1000baseT <full-duplex>)
        status: active
        nd6 options=23<PERFORMNUD,ACCEPT_RTADV,AUTO_LINKLOCAL>
```

In this example, the `em0` interface is using `fe80::21f:16ff:fe0f:275a%em0`, an auto-configured link-local address which was automatically generated from the MAC address.

Some IPv6 addresses are reserved. A list of reserved addresses can be checked in the following table:

Table 10. Example IPv6 Reserved Addresses

IPv6 address	Description	Notes
`::/128`	unspecified	Equivalent to `0.0.0.0` in IPv4.
`::1/128`	loopback address	Equivalent to `127.0.0.1` in IPv4.
`::ffff:0.0.0.0/96`	IPv4 mapped IPv6 address	The lower 32 bits are the IPv4 address for compatibility with IPv4 hosts and routers.
`fe80::/10`	link-local unicast	Equivalent to 169.254.0.0/16 in IPv4.
`fc00::/7`	unique-local	Unique local addresses are intended for local communication and are only routable within a set of cooperating sites.
`ff00::/8`	multicast	
`2000::/3`	global unicast	All global unicast addresses are assigned from this pool. The first 3 bits are `001`.
`2001:db8::/32, 3fff::/20`	documentation	IPv6 address prefix for use in documentation.

For further information on the structure of IPv6 addresses, refer to RFC4291.

7.3.4. Configuring Static IPv6 Address

To configure a FreeBSD system as an IPv6 client with a static IPv6 address it is necessary to set the IPv6 address.

Execute the following commands to meet the requirements:

```
# sysrc ifconfig_em0_ipv6="inet6 2001:db8:4672:6565:2026:5043:2d42:5344 prefixlen 64"
```

To assign a default router, specify its address executing the following command:

```
# sysrc ipv6_defaultrouter="2001:db8:4672:6565::1"
```

To configure an additional IPv6 anycast address, specify the anycast address as an `_aliasN`, as specified in rc.conf(5), followed by the `anycast` option:

```
# sysrc ifconfig_em0_alias0="inet6 2001:db8:4672:6565::a anycast"
```

Keep in mind that the applications can’t bind to anycast addresses; in that case you need to use an alias address instead.

7.3.5. Configuring Dynamic IPv6 Address

To dynamically configure the IPv6 address of the interface using SLAAC, execute the following commands:

```
# sysrc ifconfig_em0_ipv6="inet6 accept_rtadv"
# sysrc rtsold_enable="YES"
```

Note that when IPv6 packet forwarding is enabled (i.e., `ipv6_gateway_enable=YES`), the system will not configure a SLAAC address unless the `net.inet6.ip6.rfc6204w3` sysctl(8) variable is set to 1.

7.3.6. Router Advertisement and Host Auto Configuration

This section demonstrates how to setup rtadvd(8) on an IPv6 router to advertise the IPv6 network prefix and default route.

To enable rtadvd(8), execute the following command:

```
# sysrc rtadvd_enable="YES"
```

It is important to specify the interface on which to do IPv6 router advertisement. For example, to tell rtadvd(8) to use `em0`:

```
# sysrc rtadvd_interfaces="em0"
```

Next, create the configuration file, /etc/rtadvd.conf as seen in this example:

```
em0:\
    :addrs#1:addr="2001:db8:1f11:246::":prefixlen#64:tc=ether:
```

Replace `em0` with the interface to be used and `2001:db8:1f11:246::` with the prefix of the allocation.

For a dedicated `/64` subnet, nothing else needs to be changed. Otherwise, change the `prefixlen#` to the correct value.

7.3.7. IPv6 and IPv4 Address mapping

When IPv6 is enabled on a server, there may be a need to enable IPv4 mapped IPv6 address communication. This compatibility option allows for IPv4 addresses to be represented as IPv6 addresses. Permitting IPv6 applications to communicate with IPv4 and vice versa may be a security issue.

This option may not be required in most cases and is available only for compatibility. This option will allow IPv6-only applications to work with IPv4 in a dual stack environment. This is most useful for third party applications which may not support an IPv6-only environment.

To enable this feature execute the following command:

```
# sysrc ipv6_ipv4mapping="YES"
```

7.4. Wireless Networks

Most wireless networks are based on the IEEE® 802.11 standards.

FreeBSD supports networks that operate using 802.11a, 802.11b, 802.11g and 802.11n.

> 802.11ac support on FreeBSD is currently under development.

A basic wireless network consists of multiple stations communicating with radios that broadcast in either the 2.4GHz or 5GHz band, though this varies according to the locale and is also changing to enable communication in the 2.3GHz and 4.9GHz ranges.

There are three basic steps to configure a wireless network:

1. Scan and select an access point
2. Authenticate the station
3. Configure an IP address or use DHCP.

The following sections discuss each step.

7.4.1. Quick Start to Connect to a Wireless Network

Connecting FreeBSD to an existing wireless network is a very common situation.

This Quick Start procedure shows the steps required to connect to a network using basic authentication. A more detailed procedure is given in the next section.

The first step is to obtain the Service Set Identifier (SSID) and Pre-Shared Key (PSK) for the wireless network from the network administrator.

The second step is to add an entry for this network to /etc/wpa_supplicant.conf.

If the file does not exist, create it. Consult wpa_supplicant.conf(5) for details on formatting entries in this file.

```
ctrl_interface=/var/run/wpa_supplicant
eapol_version=1
```

```
ap_scan=1
fast_reauth=1

network={
 ssid="myssid" ①
 psk="mypsk" ②
}
```

① Is the SSID of the wireless network. Replace it with the name of the wireless network.

② Is the PSK of the wireless network. Replace it with the password of the wireless network.

The third step is to add the network interface entry to configure the network on startup. Obtain the network device interface by using the sysctl `net.wlan.devices`. In the listing below, the output of this sysctl shows the network device interface is "iwm0".

```
# sysctl net.wlan.devices
```

The output should be similar to the following:

```
net.wlan.devices: iwm0
```

In the sysrc line that follows, replace "iwm0" with the output of the sysctl command if needed.

```
# sysrc wlans_iwm0="wlan0"
# sysrc ifconfig_wlan0="WPA DHCP"
```

- And the last step is to restart the `netif` service executing the following command:

```
# service netif restart
```

7.4.2. Basic Wireless Configuration

This section provides a more detailed example of configuring a wireless network card. To find out what wireless network cards are in the system check the section Identify Network Adapters.

```
# ifconfig wlan0 create wlandev iwm0
```

To make the change persist across reboots execute the following command:

```
# sysrc wlans_iwm0="wlan0"
```

Since the regulatory situation is different in various parts of the world, it is necessary to correctly set the domains that apply to your location to have the correct information about what channels can be used.

The available region definitions can be found in /etc/regdomain.xml. To set the data at runtime, use `ifconfig`:

```
# ifconfig wlan0 regdomain etsi2 country AT
```

To persist the settings, add it to /etc/rc.conf:

```
# sysrc create_args_wlan0="country AT regdomain etsi2"
```

7.4.3. Scan Wireless Networks

Available wireless networks can be scanned using ifconfig(8).

To list the wireless networks execute the following command:

```
# ifconfig wlan0 up list scan
```

The output should be similar to the following:

```
SSID/MESH ID                      BSSID              CHAN RATE    S:N     INT CAPS
FreeBSD                           e8:d1:1b:1b:58:ae    1   54M  -47:-96   100 EP   RSN
BSSLOAD HTCAP WPS WME
NetBSD                            d4:b9:2f:35:fe:08    1   54M  -80:-96   100 EP   RSN
BSSLOAD HTCAP WPS WME
OpenBSD                           fc:40:09:c6:31:bd   36   54M  -94:-96   100 EPS
VHTPWRENV APCHANREP RSN WPS BSSLOAD HTCAP VHTCAP VHTOPMODE WME
GNU-Linux                         dc:f8:b9:a0:a8:e0   44   54M  -95:-96   100 EP   WPA
RSN WPS HTCAP VHTCAP VHTOPMODE WME VHTPWRENV
Windows                           44:48:b9:b3:c3:ff   44   54M  -84:-96   100 EP
BSSLOAD VHTPWRENV HTCAP WME RSN VHTCAP VHTOPMODE WPS
MacOS                             46:48:b9:b3:c3:ff   44   54M  -84:-96   100 EP
BSSLOAD VHTPWRENV HTCAP WME RSN VHTCAP VHTOPMODE WPS
```

1. SSID/MESH ID identifies the name of the network.
2. BSSID identifies the MAC address of the access point.
3. CAPS field identifies the type of each network and the capabilities of the stations operating there (see the definition of `list scan` in ifconfig(8) for more details).

7.4.4. Connection and Authentication to a Wireless Network

Once a wireless network has been selected from the list of scanned networks, it is necessary to perform the connection and the authentication. In the vast majority of wireless networks, authentication is done with a password configured in the router. Other schemes require cryptographic handshakes to be completed before data traffic can flow, either using pre-shared keys or secrets, or more complex schemes that involve backend services such as RADIUS.

Authenticate with WPA2/WPA/Personal

The authentication process in a wireless network is managed by wpa_supplicant(8).

The wpa_supplicant(8) configuration will be made in the /etc/wpa_supplicant.conf file. For more information, see wpa_supplicant.conf(5).

Once the scanning of the wireless networks has been carried out, a network has been chosen and have the password (PSK), that information will be added to the file /etc/wpa_supplicant.conf as in the following example:

```
network={
        scan_ssid=1 ①
        ssid="FreeBSD" ②
        psk="12345678" ③
}
```

① SSID scan technique. Only need to use this option if the network is hidden.

② Network name.

③ Password of the wireless network.

The next step will be to configure the wireless connection in the file /etc/rc.conf.

To use a static address it will be necessary to execute the following command:

```
# sysrc ifconfig_wlan0="inet 192.168.1.20 netmask 255.255.255.0"
```

To use a dynamic address it will be necessary to execute the following command:

```
# sysrc ifconfig_wlan0="WPA DHCP"
```

Then restart the network executing the following command:

```
# service netif restart
```

More information on how to perform more advanced methods of authentication can be obtained at Wireless Advanced Authentication.

Authenticate with Open Networks

It is important that the user is **very** careful when connecting to open networks without any kind of authentication.

Once the wireless network scan is done and the SSID of the wireless network is selected, execute the following command:

```
# ifconfig wlan0 ssid SSID
```

And then execute dhclient(8) to get the address configured:

```
# dhclient wlan0
```

7.4.5. Using Both Wired and Wireless Connections

A wired connection provides better performance and reliability, while a wireless connection provides flexibility and mobility. Laptop users typically want to roam seamlessly between the two types of connections.

On FreeBSD, it is possible to combine two or even more network interfaces together in a "failover" fashion. This type of configuration uses the most preferred and available connection from a group of network interfaces, and the operating system switches automatically when the link state changes.

Link aggregation and failover is covered in Link Aggregation and Failover and an example for using both wired and wireless connections is provided at Failover Mode Between Ethernet and Wireless Interfaces.

7.5. Hostname

The hostname represents the fully qualified domain name (FQDN) of the host on the network.

If no hostname is set for the host, FreeBSD will call itself `Amnesiac`.

7.5.1. Check The Current Hostname

hostname(1) can be used to check the current hostname:

```
$ hostname
```

The output should be similar to the following:

```
freebsdhostname.example.com
```

7.5.2. Change Hostname

To change the hostname of the host and persist it across reboots execute the following command:

```
# sysrc hostname="freebsdhostname.example.com"
```

7.6. DNS

The DNS could be understood as a telephone directory in which an IP is identified to a hostname and vice versa.

There are three files that handle how a FreeBSD system interact with the DNS. These three files are hosts(5), resolv.conf(5) and nsswitch.conf(5)

Unless otherwise stated in the /etc/nsswitch.conf file, FreeBSD will look at the addresses in the /etc/hosts file and then the DNS information in the /etc/resolv.conf file.

> The nsswitch.conf(5) file specifies how the nsdispatch (name-service switch dispatcher) should operate.
>
> By default, the hosts section of the /etc/nsswitch.conf file will be as follows:
>
> ```
> hosts: files dns
> ```
>
> For example, in case of using the nscd(8) service. The order of preference could be changed by leaving the line as follows:
>
> ```
> hosts: files cache dns
> ```

7.6.1. Local addresses

The /etc/hosts file is a simple text database who provide host name to IP address mappings. Entries for local computers connected via a LAN can be added to this file for simplistic naming purposes instead of setting up a DNS server. Additionally, /etc/hosts can be used to provide a local record of Internet names, reducing the need to query external DNS servers for commonly accessed names.

For example, in the case of having a local instance of www/gitlab-ce in a local environment, it could be added as follows to the file /etc/hosts:

```
192.168.1.150 git.example.com git
```

7.6.2. Configuring the Nameserver

How a FreeBSD system accesses the Internet Domain Name System (DNS) is controlled by resolv.conf(5).

The most common entries to /etc/resolv.conf are:

`nameserver`	The IP address of a name server the resolver should query. The servers are queried in the order listed with a maximum of three.
`search`	Search list for hostname lookup. This is normally determined by the domain of the local hostname.
`domain`	The local domain name.

A typical /etc/resolv.conf looks like this:

```
search example.com
nameserver 147.11.1.11
nameserver 147.11.100.30
```

Only one of the `search` and `domain` options should be used.

When using DHCP, dhclient(8) usually rewrites /etc/resolv.conf with information received from the DHCP server.

If the machine in which the configuration is being made is **not** a DNS server, local-unbound(8) can be used to improve DNS lookup performance.

To enable it at boot time execute the following command:

```
# sysrc local_unbound_enable="YES"
```

To start the local-unbound(8) service execute the following command:

```
# service local_unbound start
```

7.7. Troubleshooting

When troubleshooting hardware and software configurations, check the simple things first.

- Is the network cable plugged in?
- Are the network services properly configured?
- Is the firewall configured correctly?
- Is the NIC supported by FreeBSD?
- Is the router working correctly?

Before sending a bug report, always check the Hardware Notes in the FreeBSD release page, update the version of FreeBSD to the latest STABLE version, check the mailing list archives, and search the Internet.

7.7.1. Troubleshooting in Wired Networks

If the card works, yet performance is poor, read through tuning(7). Also, check the network configuration as incorrect network settings can cause slow connections.

`No route to host` messages occur if the system is unable to route a packet to the destination host. This can happen if no default route is specified or if a cable is unplugged. Check the output of `netstat -rn` and make sure there is a valid route to the host. If there is not, read Gateways and Routes.

`ping: sendto: Permission denied` error messages are often caused by a misconfigured firewall. If a firewall is enabled on FreeBSD but no rules have been defined, the default policy is to deny all traffic, even ping(8). Refer to Firewalls for more information.

7.7.2. Troubleshooting in Wireless Networks

This section describes a number of steps to help troubleshoot common wireless networking problems.

- If the access point is not listed when scanning, check that the configuration has not limited the wireless device to a limited set of channels.
- If the device cannot associate with an access point, verify that the configuration matches the settings on the access point. This includes the authentication scheme and any security protocols. Simplify the configuration as much as possible. If using a security protocol such as WPA2 or WPA, configure the access point for open authentication and no security to see if traffic will pass.
- Once the system can associate with the access point, diagnose the network configuration using tools like ping(8).
- There are many lower-level debugging tools. Debugging messages can be enabled in the 802.11 protocol support layer using wlandebug(8).

Part II: Common Tasks

Now that the basics have been covered, this part of the book discusses some frequently used features of FreeBSD. These chapters:

- Introduce popular and useful desktop applications: browsers, productivity tools, document viewers, and more.
- Introduce a number of multimedia tools available for FreeBSD.
- Explain the process of building a customized FreeBSD kernel to enable extra functionality.
- Describe the print system in detail, both for desktop and network-connected printer setups.
- Show how to run Linux applications on the FreeBSD system.

Some of these chapters recommend prior reading, and this is noted in the synopsis at the beginning of each chapter.

Chapter 8. Desktop Environments

8.1. Synopsis

While FreeBSD is popular as a server for its performance and stability, it is also well suited for day-to-day use as a desktop. With over 36000 applications available in the FreeBSD ports tree, it is straightforward to build a customized desktop that can run a wide variety of desktop applications. This chapter demonstrates how to install popular desktop environments as well as desktop applications such as web browsers, productivity software, document viewers, and financial software.

Prerequisites:

- Readers of this chapter should already understand how to install either the X Window System or Wayland on FreeBSD.
- Readers are instructed throughout this chapter to install official packages. Refer to the section on using the ports collection to build customized packages from ports.

8.2. Desktop Environments

This section describes how to install and configure some popular desktop environments on a FreeBSD system. A desktop environment can range from a simple window manager to a complete suite of desktop applications.

Table 11. Supported desktop environments

Name	License	Package
Plasma	GPL 2.0 or later	x11/kde
GNOME	GPL 2.0 or later	x11/gnome
XFCE	GPL, LGPL, BSD	x11-wm/xfce4
MATE	GPL 2.0, LGPL 2.0	x11/mate
Cinnamon	GPL 2.0 or later	x11/cinnamon
LXQT	GPL, LGPL	x11-wm/lxqt

8.2.1. Plasma

Plasma is a easy-to-use desktop environment made by the KDE community. This desktop provides a suite of applications with a consistent look and feel, a standardized menu and toolbars, keybindings, color-schemes, internationalization, and a centralized, dialog-driven desktop configuration.

More information on Plasma can be found at the Plasma homepage. For FreeBSD-specific information, consult the FreeBSD homepage at KDE.

Install Plasma meta package

To install the Plasma desktop with a basic set of KDE Applications, execute:

```
# pkg install kde
```

Additional KDE Applications are tagged with the `kde-applications` category. Search for them by executing the following command:

```
% pkg search -S origin kde-applications
```

Minimal Plasma installation

To install a minimal Plasma desktop, execute:

```
# pkg install plasma6-plasma
```

This installation is **really** minimal. The Konsole terminal emulator must be installed separately executing:

```
# pkg install konsole
```

Configure Plasma

Plasma uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of Plasma.

Enable the D-Bus service in rc.conf(5) to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start Plasma

The preferred way to start Plasma is automatically via the x11/sddm display manager. To install x11/sddm, execute:

```
# pkg install sddm
```

Enable the SDDM service in rc.conf(5) to start at system boot:

```
# sysrc sddm_enable="YES"
```

A keyboard language, in this case Spanish, can be set in SDDM with the following command:

```
# sysrc sddm_lang="es_ES"
```

Alternatively, start Plasma in an Xorg session is by manually invoking startx(1). For this to work, the following line is needed in ~/.xinitrc:

```
% echo "exec dbus-launch --exit-with-x11 ck-launch-session startplasma-x11" >
~/.xinitrc
```

Experimentally, start Plasma manually in a Wayland session by executing the following command:

```
% exec dbus-launch ck-launch-session startplasma-wayland
```

8.2.2. GNOME

GNOME is a user-friendly desktop environment. It includes a panel for starting applications and displaying status, a desktop, a set of tools and applications, and a set of conventions that make it easy for applications to cooperate and be consistent with each other.

Install GNOME meta package

To install the GNOME meta package with GNOME Desktop and Applications, execute:

```
# pkg install gnome
```

Minimal GNOME installation

To install the GNOME-lite meta package with a GNOME desktop slimmed down for only the basics, execute:

```
# pkg install gnome-lite
```

Configure GNOME

GNOME requires `/proc` to be mounted. Add this line to `/etc/fstab` to mount this file system automatically during system startup:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
proc                    /proc           procfs  rw              0       0
```

GNOME uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of GNOME.

Enable D-BUS service in `/etc/rc.conf` to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start GNOME

GNOME Display Manager is the preferred display manager for GNOME. GDM is installed as part of the GNOME package.

Enable GDM in `/etc/rc.conf` to start at system boot:

```
# sysrc gdm_enable="YES"
```

A second method to start GNOME is by manually invoking startx(1). For this to work, the following line is needed in `~/.xinitrc`:

```
% echo "exec gnome-session" > ~/.xinitrc
```

8.2.3. XFCE

XFCE is a desktop environment based on the GTK+, lightweight and provides a simple, efficient, easy-to-use desktop. It is fully configurable, has a main panel with menus, applets, and application launchers, provides a file manager and sound manager, and is themeable. Since it is fast, light, and efficient, it is ideal for older or slower machines with memory limitations.

Install XFCE

To install the XFCE meta package, execute:

```
# pkg install xfce
```

Configure XFCE

XFCE uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of XFCE.

Enable D-BUS in `/etc/rc.conf` to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start XFCE

x11/lightdm is a display manager that supports different display technologies and is a good choice as it is very lightweight, requires little memory usage, and has fast performance.

To install it, execute:

```
# pkg install lightdm lightdm-gtk-greeter
```

Enable lightdm in `/etc/rc.conf` to start at system boot:

```
# sysrc lightdm_enable="YES"
```

A second method to start XFCE is by manually invoking startx(1). For this to work, the following line is needed in `~/.xinitrc`:

```
% echo '. /usr/local/etc/xdg/xfce4/xinitrc' > ~/.xinitrc
```

8.2.4. MATE

The MATE Desktop Environment is the continuation of GNOME 2. It provides an intuitive and attractive desktop environment using traditional metaphors.

Install MATE meta package

To install the MATE meta package that includes the MATE Desktop with some extra applications such as text editor, archiver manager, etc., execute:

```
# pkg install mate
```

Minimal MATE installation

To install the MATE lite meta package with MATE desktop slimmed down for only the basics, execute:

```
# pkg install mate-base
```

Configure MATE

MATE requires `/proc` to be mounted. Add this line to `/etc/fstab` to mount this file system automatically during system startup:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
proc                    /proc           procfs  rw              0       0
```

MATE uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of MATE. Enable D-BUS in `/etc/rc.conf` to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start MATE

x11/lightdm is a display manager that supports different display technologies and is a good choice as it is very lightweight, requires little memory usage, and has fast performance.

To install it, execute:

```
# pkg install lightdm lightdm-gtk-greeter
```

Enable lightdm in `/etc/rc.conf` to start at system boot:

```
# sysrc lightdm_enable="YES"
```

A second method to start MATE is by manually invoking startx(1). For this to work, the following line is needed in `~/.xinitrc`:

```
% echo "exec dbus-launch --exit-with-x11 ck-launch-session mate-session" > ~/.xinitrc
```

8.2.5. Cinnamon

Cinnamon is a UNIX® desktop which provides advanced innovative features and a traditional user experience. The desktop layout is similar to Gnome 2. The underlying technology is forked from Gnome Shell. The emphasis is put on making users feel at home and providing them with an easy to use and comfortable desktop experience.

Install Cinnamon

To install the Cinnamon package, execute:

```
# pkg install cinnamon
```

Configure Cinnamon

Cinnamon requires `/proc` to be mounted. Add this line to `/etc/fstab` to mount this file system automatically during system startup:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
proc                    /proc           procfs  rw              0       0
```

Cinnamon uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of Cinnamon. Enable D-BUS in `/etc/rc.conf` to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start Cinnamon

x11/lightdm is a display manager that supports different display technologies and is a good choice as it is very lightweight, requires little memory usage, and has fast performance.

To install it execute:

```
# pkg install lightdm lightdm-gtk-greeter
```

Enable lightdm in `/etc/rc.conf` to start at system boot:

```
# sysrc lightdm_enable="YES"
```

A second method to start Cinnamon is by manually invoking startx(1). For this to work, the following line is needed in `~/.xinitrc`:

```
% echo "exec dbus-launch --exit-with-x11 ck-launch-session cinnamon-session" >
~/.xinitrc
```

8.2.6. LXQT

LXQt is an advanced, easy-to-use, and fast desktop environment based on Qt technologies. It has been tailored for users who value simplicity, speed, and an intuitive interface. Unlike most desktop environments, LXQt also works fine with less powerful machines.

Install LXQT

To install the LXQT meta package, execute:

```
# pkg install lxqt
```

Configure LXQT

LXQT requires `/proc` to be mounted. Add this line to `/etc/fstab` to mount this file system automatically during system startup:

```
# Device                  Mountpoint      FStype  Options         Dump    Pass#
proc                      /proc           procfs  rw              0       0
```

LXQT uses dbus-daemon(1) for a message bus and hardware abstraction. This application is automatically installed as a dependency of LXQT.

Enable D-BUS in `/etc/rc.conf` to start at system boot:

```
# sysrc dbus_enable="YES"
```

Start LXQT

The preferred LXQT display manager is x11/sddm. To install x11/sddm, execute:

```
# pkg install sddm
```

Enable SDDM service in `/etc/rc.conf` to start at system boot:

```
# sysrc sddm_enable="YES"
```

The keyboard language can be set in SDDM by running the following command (for example, for Spanish):

```
# sysrc sddm_lang="es_ES"
```

A second method to start LXQT is by manually invoking startx(1). For this to work, the following line is needed in `~/.xinitrc`:

```
% echo "exec dbus-launch --exit-with-x11 ck-launch-session startlxqt" > ~/.xinitrc
```

8.3. Browsers

This section describes how to install and configure some popular web browsers on a FreeBSD system, from full web browsers with high resource consumption to command line web browsers with reduced resource usage.

Table 12. Supported browsers

Name	License	Package	Resources Needed
Firefox	MPL 2.0	www/firefox	Heavy
Chromium	BSD-3 and others	www/chromium	Heavy
Iridium browser	BSD-3 and others	www/iridium-browser	Heavy
Falkon	MPL 2.0	www/falkon-qtonly	Heavy
Konqueror	GPL 2.0 or later	x11-fm/konqueror	Medium

Name	License	Package	Resources Needed
Gnome Web (Epiphany)	GPL 3.0 or later	www/epiphany	Medium
qutebrowser	GPL 3.0 or later	www/qutebrowser	Medium
Dillo	GPL 3.0 or later	www/dillo2	Light
Links	GPL 2.0 or later	www/links	Light
w3m	MIT	www/w3m	Light

8.3.1. Firefox

Firefox is an open source browser that features a standards-compliant HTML display engine, tabbed browsing, popup blocking, extensions, improved security, and more. Firefox is based on the Mozilla codebase.

To install the package of the latest release version of Firefox, execute:

```
# pkg install firefox
```

To instead install Firefox Extended Support Release (ESR) version, execute:

```
# pkg install firefox-esr
```

8.3.2. Chromium

Chromium is an open source browser project that aims to build a safer, faster, and more stable web browsing experience. Chromium features tabbed browsing, popup blocking, extensions, and much more. Chromium is the open source project upon which the Google Chrome web browser is based.

To install Chromium, execute:

```
# pkg install chromium
```

> The executable for Chromium is /usr/local/bin/chrome, not /usr/local/bin/chromium.
>
> Available search engines in Chromium are region-locked. Refer to the section on Using Localization to configure an appropriate locale.

8.3.3. Iridium browser

Iridium is a free, open, and libre browser modification of the Chromium code base, with privacy being enhanced in several key areas. Automatic transmission of partial queries, keywords, metrics to central services is inhibited and only occurs with consent.

To install Iridium, execute:

```
# pkg install iridium-browser
```

8.3.4. Falkon

Falkon is a new-ish and very fast QtWebEngine browser. It aims to be a lightweight web browser available on all major platforms. Falkon has all standard functions someone can expect from a web browser. It includes bookmarks, history (both also in sidebar) and tabs. Beyond that, AdBlock plugin can block ads, Click2Flash can block Flash content and SSL Manager can edit the local CA Certificates database.

To install Falkon, execute:

```
# pkg install falkon
```

8.3.5. Konqueror

Konqueror is more than a web browser as it is also a file manager and a multimedia viewer. It supports WebKit, a rendering engine used by many modern browsers including Chromium, as well as its own KHTML engine.

To install Konqueror, execute:

```
# pkg install konqueror
```

8.3.6. Gnome Web (Epiphany)

Gnome Web (Epiphany) is a web browser designed to be as lightweight and fast as possible, at the expense of many of the features found in other browsers.

To install Gnome Web (Epiphany), execute:

```
# pkg install epiphany
```

8.3.7. qutebrowser

Qutebrowser is a keyboard-focused browser with a minimal GUI. It is based on Python and PyQt5 and free software, licensed under the GPL.

To install qutebrowser, execute:

```
# pkg install qutebrowser
```

8.3.8. Dillo

Dillo aims to be a multiplatform alternative browser that is small, stable, developer-friendly, usable, fast, and extensible. This new, experimental version of Dillo is based upon FLTK toolkit, rather than GTK1, and has been substantially rewritten.

To install Dillo, execute:

```
# pkg install dillo
```

8.3.9. Links

A lynx-like web browser with text and graphics modes with many features like displaying tables, menus, etc.

To install Links, execute:

```
# pkg install links
```

8.3.10. w3m

w3m is a pager/text-based web browser. It is a similar application to Lynx, but it has several features Lynx does not have like rendering tables and rendering frames.

To install w3m, execute:

```
# pkg install w3m
```

8.4. Development tools

This section describes how to install and configure some popular development tools on a FreeBSD system.

Table 13. Supported development tools

Name	License	Package	Resources Needed
Visual Studio Code	MIT	editors/vscode	Heavy
Qt Creator	QtGPL	devel/qtcreator	Heavy
Kdevelop	GPL 2.0 or later and LGPL 2.0 or later	devel/kdevelop	Heavy
Eclipse IDE	EPL	java/eclipse	Heavy
Vim	VIM	editors/vim	Light
Neovim	Apache 2.0	editors/neovim	Light

Name	License	Package	Resources Needed
GNU Emacs	GPL 3.0 or later	editors/emacs	Light

8.4.1. Visual Studio Code

Visual Studio Code is a type of tool that combines the simplicity of a code editor with what developers need for their core edit-build-debug cycle. It provides comprehensive editing and debugging support, an extensibility model, and lightweight integration with existing tools.

To install Visual Studio Code, execute:

```
# pkg install vscode
```

8.4.2. Qt Creator

Qt Creator is a cross-platform IDE (integrated development environment) tailored to the needs of Qt developers. Functionalities included with Qt Creator are:

- code editor with C++, QML and ECMAscript support;
- rapid code navigation tools;
- static code checking and style hints when typing;
- context sensitive help;
- visual debugger;
- integrated GUI layout and forms designer.

To install Qt Creator, execute:

```
# pkg install qtcreator
```

8.4.3. kdevelop

Open source, feature-full, plugin extensible IDE for C/C++ and other programming languages. It is based on KDevPlatform and the KDE and Qt libraries, and it has been under development since 1998.

To install kdevelop, execute:

```
# pkg install kdevelop
```

8.4.4. Eclipse IDE

The Eclipse Platform is an open extensible IDE for anything and yet nothing in particular. The Eclipse Platform provides building blocks and a foundation for constructing and running integrated software-development tools. The Eclipse Platform allows tool builders to independently develop tools that integrate with other people's tools.

To install Eclipse IDE, execute:

```
# pkg install eclipse
```

8.4.5. Vim

Vim is a highly configurable text editor built to enable efficient text editing. It is an improved version of the vi editor distributed with most UNIX systems.

Vim is often called a "programmer's editor," and so useful for programming that many consider it an entire IDE. It is not just for programmers, though. Vim is perfect for all kinds of text editing, from composing email to editing configuration files.

To install Vim, execute:

```
# pkg install vim
```

8.4.6. Neovim

Neovim is an aggressive refactor of editors/vim. It represents a complete overhaul of the codebase with many sanity improvements, including sensible defaults, a built-in terminal emulator, asynchronous plugin architecture, and powerful APIs designed for speed and extensibility. It retains full compatibility with almost all Vim plugins and scripts.

To install Neovim, execute:

```
# pkg install neovim
```

8.4.7. GNU Emacs

GNU Emacs is an extensible, customizable, free/libre text editor. At its core is an interpreter for Emacs Lisp, a dialect of the Lisp programming language with extensions to support text editing.

To install GNU Emacs, execute:

```
# pkg install emacs
```

8.5. Desktop office productivity

When it comes to productivity, users often look for an office suite or an easy-to-use word processor. While some desktop environments like KDE Plasma provide an office suite, there is no default productivity package. Several office suites and graphical word processors are available for FreeBSD, regardless of the installed desktop environments.

This section demonstrates how to install the following popular productivity software and indicates if the application is resource-heavy, takes time to compile from ports, or has any major dependencies.

Table 14. Supported Desktop office productivity suites

Name	License	Package	Resources Needed
LibreOffice	MPL 2.0	editors/libreoffice	Heavy
Calligra Suite	LGPL and GPL	editors/calligra	Medium
AbiWord	GPL 2.0 or later	editors/abiword	Medium

8.5.1. LibreOffice

LibreOffice is a free software office suite developed by The Document Foundation. It is compatible with other major office suites and available on a variety of platforms. It is a rebranded fork of Apache OpenOffice and includes applications found in a complete office productivity suite: a word processor, spreadsheet, presentation manager, drawing program, database management program, and a tool for creating and editing mathematical formulæ. It is available in a number of different languages and internationalization has been extended to interfaces, spell checkers, and dictionaries. More information about LibreOffice can be found at libreoffice.org.

To install LibreOffice, execute:

```
# pkg install libreoffice
```

The LibreOffice package comes by default only in English. To have a localized version of LibreOffice it is necessary to install a language pack. For example, for the version localized in Spanish, it is necessary to install the package editors/libreoffice-es with the command:

```
# pkg install es-libreoffice
```

8.5.2. Calligra

The KDE Plasma desktop environment includes an office suite which can be installed separately from KDE Plasma. Calligra includes standard components that can be found in other office suites. Words is the word processor, Sheets is the spreadsheet program, Stage manages slide presentations, and Karbon is used to draw graphical documents.

To install Calligra, execute:

```
# pkg install calligra
```

8.5.3. AbiWord

AbiWord is a free word processing program similar in look and feel to Microsoft® Word. It is fast, contains many features, and is user-friendly.

AbiWord can import or export many file formats, including some proprietary ones like Microsoft® .rtf.

To install AbiWord, execute:

```
# pkg install abiword
```

8.6. Document Viewers

Some new document formats have gained popularity since the advent of UNIX® and the viewers they require may not be available in the base system. This section demonstrates how to install the following document viewers:

Table 15. Supported Document Viewers

Name	License	Package	Resources Needed
Okular	GPL 2.0	graphics/okular	Heavy
Evince	GPL 2.0	graphics/evince	Medium
ePDFView	GPL 2.0	graphics/epdfview	Medium
Xpdf	GPL 2.0	graphics/xpdf	light
Zathura	Zlib	graphics/zathura	light

8.6.1. Okular

Okular is a universal document viewer, part of the KDE Plasma project.

Okular combines excellent functionality with the versatility of supporting different kind of documents, like PDF, Postscript, DjVu, CHM, XPS, ePub and others.

To install Okular, execute:

```
# pkg install okular
```

8.6.2. Evince

Evince is a document viewer for multiple document formats including PDF and Postscript. Part of the GNOME project. The goal of evince is to replace document viewers such as ggv and gpdf with a single, simple application.

To install Evince, execute:

```
# pkg install evince
```

8.6.3. ePDFView

ePDFView is a lightweight PDF document viewer that only uses the Gtk+ and Poppler libraries. The aim of ePDFView is to make a simple PDF document viewer, similar to Evince but without using the GNOME libraries.

To install ePDFView, execute:

```
# pkg install epdfview
```

8.6.4. Xpdf

For users that prefer a small FreeBSD PDF viewer, Xpdf provides a light-weight and efficient viewer which requires few resources. It uses the standard X fonts and does not require any additional toolkit.

To install Xpdf, execute:

```
# pkg install xpdf
```

8.6.5. Zathura

Zathura is a highly customizable and functional document viewer. It provides a minimalistic and space saving interface as well as an easy usage that mainly focuses on keyboard interaction.

To install zathura, with PDF support, execute:

```
# pkg install zathura zathura-pdf-mupdf
```

Additionally, one can install graphics/zathura-pdf-poppler for alternative PDF support, graphics/zathura-ps for PostScript support, graphics/zathura-djvu for DjVu support, and graphics/zathura-cb for comic book support.

8.7. Finance

For managing personal finances on a FreeBSD desktop, some powerful and easy-to-use applications can be installed. Some are compatible with widespread file formats, such as the formats used by Quicken and Excel.

This section covers these programs:

Table 16. Supported Finance programs

Name	License	Package	Resources Needed
KMyMoney	GPL 2.0	finance/kmymoney	Heavy
GnuCash	GPL 2.0 and GPL 3.0	finance/gnucash	Heavy

8.7.1. KMyMoney

KMyMoney is a personal finance application created by the KDE community. KMyMoney aims to provide the important features found in commercial personal finance manager applications. It also highlights ease-of-use and proper double-entry accounting among its features. KMyMoney imports from standard Quicken QIF files, tracks investments, handles multiple currencies, and provides a wealth of reports.

To install KMyMoney, execute:

```
# pkg install kmymoney
```

8.7.2. GnuCash

GnuCash is part of the GNOME effort to provide user-friendly, yet powerful, applications to end-users. GnuCash can be used to keep track of income and expenses, bank accounts, and stocks. It features an intuitive interface while remaining professional.

GnuCash provides a smart register, a hierarchical system of accounts, and many keyboard accelerators and auto-completion methods. It can split a single transaction into several more detailed pieces. GnuCash can import and merge Quicken QIF files. It also handles most international date and currency formats.

To install GnuCash, execute:

```
# pkg install gnucash
```

Chapter 9. Multimedia

9.1. Synopsis

The multimedia chapter provides an overview of multimedia support on FreeBSD. Multimedia applications and technologies have become an integral part of modern computing, and FreeBSD provides robust and reliable support for a wide range of multimedia hardware and software. This chapter covers various multimedia components such as audio, video, and image processing. It also discusses various media formats and codecs, as well as tools and applications for multimedia creation and playback. Additionally, the chapter covers multimedia system configuration, troubleshooting, and optimization. FreeBSD provides a robust platform for multimedia work, suitable for both multimedia enthusiasts and professional content creators. This chapter aims to help get the most out of FreeBSD's multimedia capabilities, providing useful information and practical examples to help get started.

9.2. Setting Up the Sound Card

By default, FreeBSD will automatically detect the sound card used by the system. FreeBSD supports a wide variety of sound cards. The list of supported sound cards can be consulted in sound(4).

It is only necessary to load the sound card module if FreeBSD has not detected it correctly.

Where it is not known which sound card the system has, or which module to use, the `snd_driver` metadriver can be loaded by executing the following command:

```
# kldload snd_driver
```

Alternatively, to load the driver as a module at boot time, place the following line in /boot/loader.conf:

```
snd_driver_load="YES"
```

9.2.1. Testing Sound

To confirm the sound card is detected the following command can be executed:

```
% dmesg | grep pcm
```

The output should be similar to the following:

```
pcm0: <Conexant CX20561 (Hermosa) (Analog 2.0+HP/2.0)> at nid 26,22 and 24 on hdaa0
```

```
pcm1: <Conexant CX20561 (Hermosa) (Internal Analog Mic)> at nid 29 on hdaa0
```

The status of the sound card may also be checked using this command:

```
# cat /dev/sndstat
```

The output should be similar to the following:

```
Installed devices:
pcm0: <Conexant CX20561 (Hermosa) (Analog 2.0+HP/2.0)> (play/rec) default
pcm1: <Conexant CX20561 (Hermosa) (Internal Analog Mic)> (rec)
```

If no `pcm` devices are listed, double-check that the correct device driver was loaded. If all goes well, the sound card should now work in FreeBSD.

beep(1) can be used to produce some noise, confirming that the sound card is working:

```
% beep
```

9.2.2. Mixer

FreeBSD has different utilities to set and display sound card mixer values built on the FreeBSD Sound System:

Table 17. Supported mixer packages

Name	License	Package	Toolkit
mixer(8)	BSD-2	Included in base system	CLI
dsbmixer	BSD-2	audio/dsbmixer	Qt
KDE Plasma audio widget	GPL 2.0	audio/plasma6-plasma-pa	Qt
mixertui	BSD-2	audio/mixertui	TUI

9.2.3. Graphics Card Sound

Graphics cards often come with their own integrated sound devices, and it may be unclear which is being used as the default device. To confirm, run dmesg and look for the pcm entries to identify how the system is enumerating the outputs. Execute the following command:

```
% dmesg | grep pcm
```

The output looks something like this:

```
pcm0: <HDA NVIDIA (Unknown) PCM #0 DisplayPort> at cad 0 nid 1 on hdac0
pcm1: <HDA NVIDIA (Unknown) PCM #0 DisplayPort> at cad 1 nid 1 on hdac0
pcm2: <HDA NVIDIA (Unknown) PCM #0 DisplayPort> at cad 2 nid 1 on hdac0
pcm3: <HDA NVIDIA (Unknown) PCM #0 DisplayPort> at cad 3 nid 1 on hdac0
hdac1: HDA Codec #2: Realtek ALC889
pcm4: <HDA Realtek ALC889 PCM #0 Analog> at cad 2 nid 1 on hdac1
pcm5: <HDA Realtek ALC889 PCM #1 Analog> at cad 2 nid 1 on hdac1
pcm6: <HDA Realtek ALC889 PCM #2 Digital> at cad 2 nid 1 on hdac1
pcm7: <HDA Realtek ALC889 PCM #3 Digital> at cad 2 nid 1 on hdac1
```

The graphics card (NVIDIA®) has been enumerated before the sound card (Realtek®), with the sound card appearing as `pcm4`. The system can be configured to use the sound card as the default device by executing the following command:

```
# sysctl hw.snd.default_unit=4
```

To make this change permanent add the next line to /etc/sysctl.conf:

```
hw.snd.default_unit=4
```

9.2.4. Automatically Switching to Headphones

Some systems may struggle with switching between audio outputs, but fortunately FreeBSD allows automatic switchover to be configured in device.hints.

Identify how the system is enumerating the audio outputs by executing the following command:

```
% dmesg | grep pcm
```

The output looks something like this:

```
pcm0: <Realtek ALC892 Analog> at nid 23 and 26 on hdaa0
pcm1: <Realtek ALC892 Right Analog Headphones> at nid 22 on hdaa0
```

Add the following lines to /boot/device.hints:

```
hint.hdac.0.cad0.nid22.config="as=1 seq=15 device=Headphones"
hint.hdac.0.cad0.nid26.config="as=2 seq=0 device=speakers"
```

Keep in mind that these values are for the example indicated above. They may vary depending on the system.

9.2.5. Resampling Quality

The process of changing the sample rate of an audio stream is called resampling.

This usually happens automatically in situations like recording audio with a DI (Digital Interface) or simply when playing back audio files with very low resolution.

Example: DI uses 192000 kHz, application only accepts 48000 kHz (or the other way around).

If possible, this should be adjusted outside the sound device driver, but for cases where this is not possible (Or the hardware is bad at resampling), you have influence over the quality that is used for resampling.

Should you be unsure if you need this, for normal music listening, doing meetings and so on the default value is totally fine and most applications use 48000 kHz anyway, so resampling rarely happens.

When making music, recording sounds otherwise where quality matters or you are an audiophile, you can increase the resampling quality.

For the highest possible resample quality, execute the following command:

```
# sysctl hw.snd.feeder_rate_quality=4①
# sysctl hw.snd.feeder_rate_round=0②
```

① 100dB stopband, 164 taps and 97% bandwidth

② Large prime divisions for high accuracy

To keep these values after reboots, they can be added to the `/etc/sysctl.conf`.

It is important to keep in mind that adjusting the resample quality has influence on CPU usage and delay.

More details about resampling can be read in sound(4)

9.2.6. Troubleshooting Sound

Some common error messages and their solutions:

Table 18. Common Error Messages

Error	**Solution**
`xxx: can't open /dev/dsp!`	Type `fstat | grep dsp` to check if another application is holding the device open. Noteworthy troublemakers are esound and KDE's sound support.

Programs using audio/pulseaudio might need to restart the audio/pulseaudio daemon for the changes in `hw.snd.default_unit` to take effect. Alternatively, audio/pulseaudio settings can be changed on the fly. pacmd(1) opens a command line connection to the audio/pulseaudio daemon:

```
# pacmd
Welcome to PulseAudio 14.2! Use "help" for usage information.
>>>
```

The following command changes the default sink to card number 4 as in the previous example:

```
set-default-sink 4
```

Do not use the `exit` command to exit the command line interface. That will kill the audio/pulseaudio daemon. Use `Ctrl` + `D` instead.

9.3. Audio players

This section introduces some of the software available from the FreeBSD Ports Collection which can be used for audio playback.

Table 19. Audio players packages

Name	License	Package	Toolkit
Elisa	LGPL 3.0	audio/elisa	Qt
GNOME Music	GPL 2.0	audio/gnome-music	GTK+
Audacious	BSD-2	multimedia/audacious	Qt
cmus — C* Music Player	GPL 2.0	audio/cmus	TUI

9.3.1. Elisa

Elisa is a music player developed by the KDE community that strives to be simple and nice to use.

To install Elisa, execute:

```
# pkg install elisa
```

9.3.2. GNOME Music

GNOME Music is the new GNOME music playing application. It aims to combine an elegant and immersive browsing experience with simple and straightforward controls.

To install GNOME Music, execute:

```
# pkg install gnome-music
```

9.3.3. Audacious

Audacious is an open source audio player. As a descendant of XMMS, it provides music playback while maintaining efficient use of system resources.

To install Audacious, execute:

```
# pkg install audacious-qt6 audacious-plugins-qt6
```

Audacious supports OSS natively, but must be configured in the settings on the Audio tab.

9.3.4. cmus — C* Music Player

cmus is a small, fast and powerful console music player for Unix-like operating systems.

Beside various input and output plugins, it has support for CUE sheets, uses Unicode, supports scrobbling (to e.g. last.fm or libre.fm), can run external programs on selected files like tag-editors, can be controlled via UNIX sockets and many more.

To install cmus, execute:

```
# pkg install cmus
```

9.4. Video players

This section introduces some of the software available from the FreeBSD Ports Collection which can be used for video playback.

Table 20. Video players packages

Name	License	Package	Toolkit
mpv	GPL 2.0	multimedia/mpv	CLI
SMPlayer	GPL 2.0	multimedia/smplayer	Qt
VLC media player	GPL 2.0	multimedia/vlc	Qt
Kodi (XBMC)	GPL 2.0	multimedia/kodi	X11

9.4.1. mpv

mpv is a free (as in freedom) media player for the command line. It supports a wide variety of media file formats, audio and video codecs, and subtitle types. mpv can leverage most hardware decoding APIs on all platforms.

Also targeted towards vidoephile, it has several high quality up-/downscaling algorithms which can also be, like all settings, controlled via profiles. Profiles can be selected via command-line switches or based on conditions like the folder name.

It also supports high quality HDR to SDR conversion with fine-grained control, enabling everyone to enjoy HDR content on any display.

To install mpv, execute:

```
# pkg install mpv
```

Here is a short sample ~/.config/mpv/mpv.conf

```
vo=gpu-next①
gpu-context=wayland②
hwdec=auto③
screenshot-jpeg-quality=100
scale=ewa_lanczos4sharpest④
dscale=ewa_lanczos4sharpest

[TTRPG]⑤
profile-cond=path:find('ttrpg')⑥
profile=fast⑦
save-position-on-quit=yes

[hdr]⑧
scale=bilinear
dscale=bilinear
dither=no
correct-downscaling=no
linear-downscaling=no
sigmoid-upscaling=no
hdr-compute-peak=yes
hdr-peak-percentile=99.995
hdr-contrast-recovery=0.30
```

① Use the new gpu-next video output. A lot of features need this and/or provide higher quality when enabled

② Wayland users should manually set this when seeing graphical glitches

③ Automatically detect if GPU hardware decoding is available and uses it

④ Best scaling algorithm. Disable if your hardware is not fast enough and videos start to skip frames

⑤ Example profile for long videos of recorded TTRPG sessions

⑥ Only apply when the path contains ttrpg

⑦ Use the builtin fast profile for lowest possible CPU/GPU usage

⑧ Example profile for users with a slow iGPU that still want to enjoy HDR content

Example how a user with a iGPU would be able to watch HDR content with this sample config

```
$ mpv --profile=hdr hdr_video.mkv
```

For more examples on how mpv can be used, consult the mpv reference manual.

9.4.2. SMPlayer

SMPlayer intends to be a complete front-end for MPlayer, from basic features like playing videos, DVDs, and VCDs to more advanced features like support for MPlayer filters and more.

To install SMPlayer, execute:

```
# pkg install smplayer
```

9.4.3. VLC media player

VLC media player is a highly portable multimedia player for various audio and video formats (MPEG-1, MPEG-2, MPEG-4, DivX, mp3, ogg, and more) as well as DVD's, VCD's, and various streaming protocols. It can also be used as a server to stream in unicast or multicast in IPv4 or IPv6 on a high-bandwidth network. VLC also has the ability to transcode media on-the-fly for streaming or saving to disk.

To install VLC, execute:

```
# pkg install vlc
```

9.4.4. Kodi (XBMC)

Kodi (formerly known as XBMC) is a free and open source cross-platform media-player and entertainment hub. It allows users to play and view most videos, music, podcasts, and other digital media files from local and network storage media and the internet.

To install Kodi, execute:

```
# pkg install kodi
```

9.5. Conferencing and Meetings

A FreeBSD desktop environment can be used to join video conferences. This section will explain how to configure the webcam and which videoconferencing applications are supported on FreeBSD.

9.5.1. Setting Up the Webcam

To allow FreeBSD access to the webcam and perform its configuration it is necessary to install certain utilities:

- multimedia/webcamd is a daemon that enables the use of hundreds of different USB based webcam and DVB USB devices.
- multimedia/pwcview is an application that can be used to view the video stream of the webcam.

To install the required utilities, execute:

```
# pkg install webcamd pwcview
```

Enable the webcamd(8) service in `/etc/rc.conf` to start it at system boot:

```
# sysrc webcamd_enable=YES
```

The user must belong to the `webcamd` group. To add the user to `webcamd` group execute the following command:

```
# pw groupmod webcamd -m username
```

Since multimedia/webcamd needs the cuse(3) module this module must be loaded by executing the following command:

```
# kldload cuse
```

To load cuse(3) at system boot, execute the command:

```
# sysrc kld_list+=cuse
```

Once the utilities have been installed the list of available webcams can be shown with webcamd(8):

```
# webcamd -l
```

The output should be similar to the following:

```
webcamd [-d ugen0.2] -N SunplusIT-Inc-HP-TrueVision-HD-Camera -S unknown -M 0 ①
webcamd [-d ugen1.3] -N Realtek-802-11n-WLAN-Adapter -S 00e04c000001 -M 0
```

① Available webcam

Configure the available webcam executing the following command:

```
# sysrc webcamd_0_flags="-N SunplusIT-Inc-HP-TrueVision-HD-Camera" ①
```

Note here that if this is a plug-and-play USB webcam, changing the USB port to which it is connected will change the output from `webcamd -l`, specifically the device identifier, and the entry in rc.conf might need to be updated. To avoid this issue either use the device name (`-N` option) and/or match with the device serial number if known (`-S` option) as displayed in the output of `webcamd -l`. For static devices, for example integrated laptop cameras, you can use the device identifier (`-d` option).

The webcamd(8) service must be started by executing the following command:

```
# service webcamd start
```

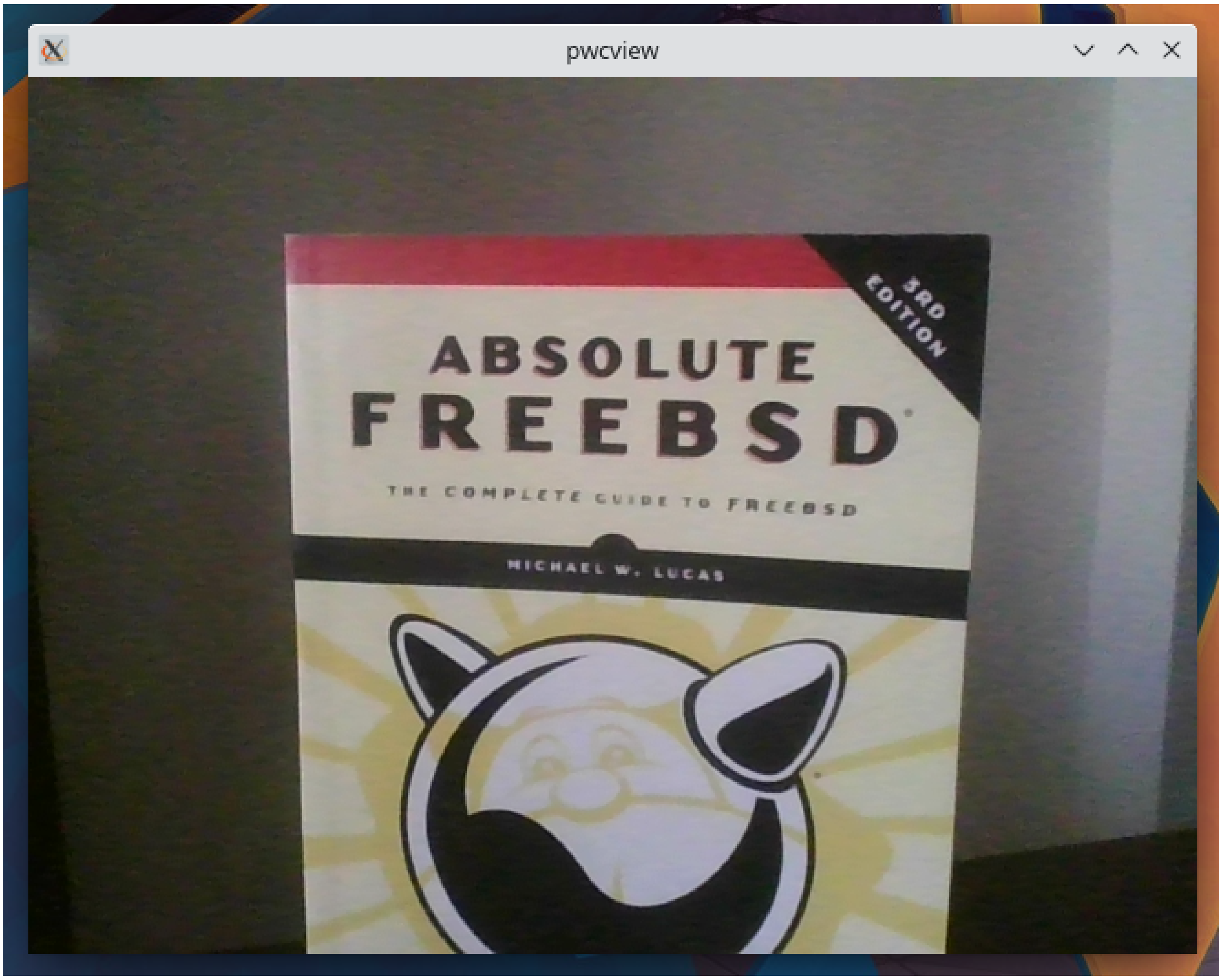
The output should be similar to the following:

```
Starting webcamd.
webcamd 1616 - - Attached to ugen0.2[0]
```

multimedia/pwcview can be used to check the proper functioning of the webcam. The following command can be used to execute multimedia/pwcview:

```
% pwcview -f 30 -s vga
```

Then multimedia/pwcview will display the webcam:


pwcview
ABSOLUTE FREEBSD
THE COMPLETE GUIDE TO FREEBSD
3RD EDITION
MICHAEL W. LUCAS

9.5.2. Meetings software status

FreeBSD currently supports the following tools used to carry out videoconferences.

Table 21. Meeting software

Name	Firefox status	Chromium status	Website
Microsoft Teams	Does not work	Works	https://teams.live.com
Google Meet	Works	Works	https://meet.google.com/
Zoom	Works	Works	https://zoom.us
Jitsi	Does not work	Works	https://meet.jit.si/
BigBlueButton	Does not work	Works	https://bigbluebutton.org/

9.6. Image Scanners

In FreeBSD, access to image scanners is provided by SANE (Scanner Access Now Easy), which is available in the FreeBSD Ports Collection.

9.6.1. Checking the Scanner

Before attempting any configuration it is important to check the scanner is supported by SANE.

With the scanner connected, run the following command to get all connected USB devices:

```
# usbconfig list
```

The output should be similar to the following:

```
ugen4.2: <LITE-ON Technology USB NetVista Full Width Keyboard.> at usbus4, cfg=0
md=HOST spd=LOW (1.5Mbps) pwr=ON (70mA)
ugen4.3: <Logitech USB Optical Mouse> at usbus4, cfg=0 md=HOST spd=LOW (1.5Mbps)
pwr=ON (100mA)
ugen3.2: <HP Deskjet 1050 J410 series> at usbus3, cfg=0 md=HOST spd=HIGH (480Mbps)
pwr=ON (2mA)
```

Run the following command to obtain the `idVendor` and the `idProduct`:

```
# usbconfig -d 3.2 dump_device_desc
```

Note here that the scanner is a plug-and-play device, and changing the USB port to which it is connected will change the output from `usbconfig list`.

The output should be similar to the following:

```
ugen3.2: <HP Deskjet 1050 J410 series> at usbus3, cfg=0 md=HOST spd=HIGH (480Mbps)
pwr=ON (2mA)

bLength = 0x0012
bDescriptorType = 0x0001
bcdUSB = 0x0200
bDeviceClass = 0x0000  <Probed by interface class>
bDeviceSubClass = 0x0000
bDeviceProtocol = 0x0000
bMaxPacketSize0 = 0x0040
idVendor = 0x03f0
idProduct = 0x8911
bcdDevice = 0x0100
iManufacturer = 0x0001  <HP>
iProduct = 0x0002  <Deskjet 1050 J410 series>
bNumConfigurations = 0x0001
```

Once the `idVendor` and the `idProduct` have been obtained, it is necessary to check in the list of supported devices of SANE if the scanner is supported by filtering by the idProduct.

9.6.2. SANE Configuration

SANE provides the access to the scanner via backends. To be able to scan with FreeBSD the graphics/sane-backends package must be installed by running the following command:

```
# pkg install sane-backends
```

Some USB scanners require firmware to be loaded. Like the HP scanner used in the example above, which needs the package print/hplip installed.

After installing the necessary packages devd(8) must be configured to allow FreeBSD access to the scanner.

Add the `saned.conf` file to /usr/local/etc/devd/saned.conf with the following content:

```
notify 100 {
        match "system" "USB";
        match "subsystem" "INTERFACE";
        match "type" "ATTACH";
        match "cdev" "ugen[0-9].[0-9]";
        match "vendor" "0x03f0"; ①
        match "product" "0x8911"; ②
        action "chown -L cups:saned /dev/\$cdev && chmod -L 660 /dev/\$cdev";
};
```

① `vendor`: Is the idVendor obtained previously by running the `usbconfig -d 3.2 dump_device_desc` command.

② `product`: Is the idProduct obtained previously by running the `usbconfig -d 3.2 dump_device_desc` command.

After that devd(8) must be restarted by running the following command:

```
# service devd restart
```

The SANE backends include scanimage(1) which can be used to list the devices and perform an image acquisition.

Execute scanimage(1) with `-L` argument to list the scanner devices:

```
# scanimage -L
```

The output should be similar to the following:

```
device `hpaio:/usb/Deskjet_1050_J410_series?serial=XXXXXXXXXXXXXX' is a Hewlett-
```

```
Packard Deskjet_1050_J410_series all-in-one
```

If scanimage(1) is not able to identify the scanner, this message will appear:

```
No scanners were identified. If you were expecting something different,
check that the scanner is plugged in, turned on and detected by the
sane-find-scanner tool (if appropriate). Please read the documentation
which came with this software (README, FAQ, manpages).
```

Once scanimage(1) sees the scanner, the configuration is complete and the scanner is now ready to use.

To activate the service and have it run at boot execute the following command:

```
# sysrc saned_enable=YES
```

While scanimage(1) can be used to perform an image acquisition from the command line, it is often preferable to use a graphical interface to perform image scanning.

Table 22. Graphical scanner programs

Name	**License**	**Package**
skanlite	GPL 2.0	graphics/skanlite
GNOME Simple Scan	GPL 3.0	graphics/simple-scan
XSANE	GPL 2.0	graphics/xsane

Chapter 10. Configuring the FreeBSD Kernel

10.1. Synopsis

The kernel is the core of the FreeBSD operating system. It is responsible for managing memory, enforcing security controls, networking, disk access, and much more. While much of FreeBSD is dynamically configurable, some users may wish to configure and compile a custom kernel.

Read this chapter to learn:

- When to build a custom kernel.
- How to take a hardware inventory.
- How to customize a kernel configuration file.
- How to use the kernel configuration file to create and build a new kernel.
- How to install the new kernel.
- How to troubleshoot if things go wrong.

All of the commands listed in the examples in this chapter should be executed as `root`.

10.2. Why Build a Custom Kernel?

Traditionally, FreeBSD used a monolithic kernel. The kernel was one large program, supported a fixed list of devices, and in order to change the kernel's behavior, one had to compile and then reboot into a new kernel.

Today, most of the functionality in the FreeBSD kernel is contained in modules which can be dynamically loaded and unloaded from the kernel as necessary. This allows the running kernel to adapt immediately to new hardware and for new functionality to be brought into the kernel. This is known as a modular kernel.

Occasionally, it is still necessary to perform static kernel configuration. Sometimes the needed functionality is so tied to the kernel that it can not be made dynamically loadable. Some security environments prevent the loading and unloading of kernel modules and require that only needed functionality is statically compiled into the kernel.

Building a custom kernel is often a rite of passage for advanced BSD users. This process, while time consuming, can provide benefits to the FreeBSD system. Unlike the GENERIC kernel, which must support a wide range of hardware, a custom kernel can be stripped down to only provide support for that computer's hardware. This has a number of benefits, such as:

- Faster boot time. Since the kernel will only probe the hardware on the system, the time it takes the system to boot can decrease.
- Lower memory usage. A custom kernel often uses less memory than the GENERIC kernel by omitting unused features and device drivers. This is important because the kernel code remains resident in physical memory at all times, preventing that memory from being used by applications. For this reason, a custom kernel is useful on a system with a small amount of RAM.

- Additional hardware support. A custom kernel can add support for devices which are not present in the GENERIC kernel.

> When building a custom kernel, it is important to note that non-default configurations are less thoroughly tested than the GENERIC configuration. While customizing the kernel can provide specific benefits it also increases the risk of encountering build or runtime issues. Custom kernel configurations are recommended only for advanced users who have a specific reason for making changes and are willing to engage in the debugging process if necessary.

Before building a custom kernel, consider the reason for doing so. If there is a need for specific hardware support, it may already exist as a module.

Kernel modules exist in /boot/kernel and may be dynamically loaded into the running kernel using kldload(8). Most kernel drivers have a loadable module and manual page. For example, the ath(4) wireless network driver has the following information in its manual page:

```
Alternatively, to load the driver as a module at boot time, place the
following line in loader.conf(5):

    if_ath_load="YES"
```

Adding `if_ath_load="YES"` to /boot/loader.conf will load this module dynamically at boot time.

In some cases, there is no associated module in /boot/kernel. This is mostly true for certain subsystems.

10.3. Finding the System Hardware

Before editing the kernel configuration file, it is recommended to perform an inventory of the machine’s hardware. On a dual-boot system, the inventory can be created from the other operating system. For example, Microsoft®'s Device Manager contains information about installed devices.

> Some versions of Microsoft® Windows® have a System icon which can be used to access Device Manager.

If FreeBSD is the only installed operating system, use dmesg(8) to determine the hardware that was found and listed during the boot probe. Most device drivers on FreeBSD have a manual page which lists the hardware supported by that driver. For example, the following lines indicate that the psm(4) driver found a mouse:

```
psm0: <PS/2 Mouse> irq 12 on atkbdc0
psm0: [GIANT-LOCKED]
psm0: [ITHREAD]
psm0: model Generic PS/2 mouse, device ID 0
```

Since this hardware exists, this driver should not be removed from a custom kernel configuration file.

If the output of `dmesg` does not display the results of the boot probe output, instead read the contents of /var/run/dmesg.boot.

Another tool for finding hardware is pciconf(8), which provides more verbose output. For example:

```
% pciconf -lv
ath0@pci0:3:0:0:        class=0x020000 card=0x058a1014 chip=0x1014168c rev=0x01 hdr
=0x00
    vendor     = 'Atheros Communications Inc.'
    device     = 'AR5212 Atheros AR5212 802.11abg wireless'
    class      = network
    subclass   = ethernet
```

This output shows that the ath driver located a wireless Ethernet device.

The `-k` flag of man(1) can be used to provide useful information. For example, it can be used to display a list of manual pages which contain a particular device brand or name:

```
# man -k Atheros
ath(4)                       - Atheros IEEE 802.11 wireless network driver
ath_hal(4)                   - Atheros Hardware Access Layer (HAL)
```

Once the hardware inventory list is created, refer to it to ensure that drivers for installed hardware are not removed as the custom kernel configuration is edited.

10.4. The Configuration File

In order to create a custom kernel configuration file and build a custom kernel, the full FreeBSD source tree must first be installed.

If /usr/src/ does not exist or it is empty, source has not been installed. Source can be installed with Git using the instructions in "Using Git".

Once source is installed, review the contents of /usr/src/sys. This directory contains a number of subdirectories, including those which represent the following supported architectures: amd64, i386, powerpc, and sparc64. Everything inside a particular architecture's directory deals with that architecture only and the rest of the code is machine independent code common to all platforms. Each supported architecture has a conf subdirectory which contains the GENERIC kernel configuration file for that architecture.

Do not make edits to GENERIC. Instead, copy the file to a different name and make edits to the copy. The convention is to use a name with all capital letters. When maintaining multiple FreeBSD machines with different hardware, it is a good idea to name it after the machine's hostname. This example creates a copy, named MYKERNEL, of the GENERIC configuration file for the `amd64` architecture:

```
# cd /usr/src/sys/amd64/conf
# cp GENERIC MYKERNEL
```

MYKERNEL can now be customized with any `ASCII` text editor. The default editor is vi, though an easier editor for beginners, called ee, is also installed with FreeBSD.

The format of the kernel configuration file is simple. Each line contains a keyword that represents a device or subsystem, an argument, and a brief description. Any text after a `#` is considered a comment and ignored. To remove kernel support for a device or subsystem, put a `#` at the beginning of the line representing that device or subsystem. Refrain from adding or removing a `#` for any line you do not fully understand.

> ⚠ It is easy to remove support for a device or option and end up with a broken kernel. For example, if the ata(4) driver is removed from the kernel configuration file, a system using `ATA` disk drivers may not boot. When in doubt, just leave support in the kernel.

In addition to the brief descriptions provided in this file, additional descriptions are contained in NOTES, which can be found in the same directory as GENERIC for that architecture. For architecture independent options, refer to /usr/src/sys/conf/NOTES.

> When finished customizing the kernel configuration file, save a backup copy to a location outside of /usr/src.
>
> Alternately, keep the kernel configuration file elsewhere and create a symbolic link to the file:
>
> ```
> # cd /usr/src/sys/amd64/conf
> # mkdir /root/kernels
> # cp GENERIC /root/kernels/MYKERNEL
> # ln -s /root/kernels/MYKERNEL
> ```

An `include` directive is available for use in configuration files. This allows another configuration file to be included in the current one, making it easy to maintain small changes relative to an existing file. If only a small number of additional options or drivers are required, this allows a delta to be maintained with respect to GENERIC, as seen in this example:

```
include GENERIC
ident MYKERNEL

options         IPFIREWALL
options         DUMMYNET
options         IPFIREWALL_DEFAULT_TO_ACCEPT
options         IPDIVERT
```

Using this method, the local configuration file expresses local differences from a GENERIC kernel.

As upgrades are performed, new features added to GENERIC will also be added to the local kernel unless they are specifically prevented using `nooptions` or `nodevice`. A comprehensive list of configuration directives and their descriptions may be found in config(5).

> To build a file which contains all available options, run the following command as `root`:
>
> ```
> # cd /usr/src/sys/arch/conf && make LINT
> ```

10.5. Building and Installing a Custom Kernel

Once the edits to the custom configuration file have been saved, the source code for the kernel can be compiled using the following steps:

Procedure: Building a Kernel

1. Change to this directory:

   ```
   # cd /usr/src
   ```

2. Compile the new kernel by specifying the name of the custom kernel configuration file:

   ```
   # make buildkernel KERNCONF=MYKERNEL
   ```

3. Install the new kernel associated with the specified kernel configuration file. This command will copy the new kernel to /boot/kernel/kernel and save the old kernel to /boot/kernel.old/kernel:

   ```
   # make installkernel KERNCONF=MYKERNEL
   ```

4. Shutdown the system and reboot into the new kernel. If something goes wrong, refer to The kernel does not boot.

By default, when a custom kernel is compiled, all kernel modules are rebuilt. To update a kernel faster or to build only custom modules, edit /etc/make.conf before starting to build the kernel.

For example, this variable specifies the list of modules to build instead of using the default of building all modules:

```
MODULES_OVERRIDE = linux acpi
```

Alternately, this variable lists which modules to exclude from the build process:

```
WITHOUT_MODULES = linux acpi sound
```

This variable instructs the build to skip kernel modules and compile only the kernel itself:

```
NO_MODULES = yes
```

The Ports Framework includes drivers and other modules that depend on kernel interfaces, such as graphics/drm-kmod or emulators/virtualbox-ose-kmod. With the `PORTS_MODULES` variable, every time the kernel is built, the ports containing kernel modules are re-built against the updated sources. This ensures the kernel module stays in-sync with the kernel itself. The kernel and ports trees should be updated together for maximum compatibility. `PORTS_MODULES` can be added to /etc/make.conf to ensure all kernels you build also rebuild these modules.

```
PORTS_MODULES = drm-kmod virtualbox-ose-kmod
```

Additional variables are available. Refer to make.conf(5) for details.

10.6. If Something Goes Wrong

There are four categories of trouble that can occur when building a custom kernel:

`config` fails

If `config` fails, it will print the line number that is incorrect. As an example, for the following message, make sure that line 17 is typed correctly by comparing it to GENERIC or NOTES:

```
config: line 17: syntax error
```

`make` fails

If `make` fails, it is usually due to an error in the kernel configuration file which is not severe enough for `config` to catch. Review the configuration, and if the problem is not apparent, send an email to the FreeBSD general questions mailing list which contains the kernel configuration file.

The kernel does not boot

If the new kernel does not boot or fails to recognize devices, do not panic! Fortunately, FreeBSD has an excellent mechanism for recovering from incompatible kernels. Simply choose the kernel to boot from at the FreeBSD boot loader. This can be accessed when the system boot menu appears by selecting the "Escape to a loader prompt" option. At the prompt, type `boot kernel.old`, or the name of any other kernel that is known to boot properly.

After booting with a good kernel, check over the configuration file and try to build it again. One helpful resource is /var/log/messages which records the kernel messages from every successful boot. Also, dmesg(8) will print the kernel messages from the current boot.

When troubleshooting a kernel make sure to keep a copy of a kernel that is known to work, such as GENERIC. This is important because every time a new kernel is installed, kernel.old is overwritten with the last installed kernel, which may or may not be bootable. As soon as possible, move the working kernel by renaming the directory containing the good kernel:

```
# mv /boot/kernel /boot/kernel.bad
# mv /boot/kernel.good /boot/kernel
```

The kernel works, but ps(1) does not

If the kernel version differs from the one that the system utilities have been built with, for example, a kernel built from -CURRENT sources is installed on a -RELEASE system, many system status commands like ps(1) and vmstat(8) will not work. To fix this, recompile and install a world built with the same version of the source tree as the kernel. It is never a good idea to use a different version of the kernel than the rest of the operating system.

Chapter 11. Printing

Putting information on paper is a vital function, despite many attempts to eliminate it. Printing has two basic components. The data must be delivered to the printer, and must be in a form that the printer can understand.

11.1. Quick Start

Basic printing can be set up quickly. The printer must be capable of printing plain `ASCII` text. For printing to other types of files, see Filters.

1. Create a directory to store files while they are being printed:

   ```
   # mkdir -p /var/spool/lpd/lp
   # chown daemon:daemon /var/spool/lpd/lp
   # chmod 770 /var/spool/lpd/lp
   ```

2. As `root`, create /etc/printcap with these contents:

   ```
   lp:\
   lp=/dev/unlpt0:\  ①
   sh:\
   mx#0:\
   sd=/var/spool/lpd/lp:\
   lf=/var/log/lpd-errs:
   ```

   ① This line is for a printer connected to a `USB` port.
   For a printer connected to a parallel or "printer" port, use:

   ```
   :lp=/dev/lpt0:\
   ```

   For a printer connected directly to a network, use:

   ```
   :lp=:rm=network-printer-name:rp=raw:\
   ```

   Replace *network-printer-name* with the `DNS` host name of the network printer.

3. Enable LPD by editing /etc/rc.conf, adding this line:

   ```
   lpd_enable="YES"
   ```

   Start the service:

```
# service lpd start
Starting lpd.
```

4. Print a test:

```
# printf "1. This printer can print.\n2. This is the second line.\n" | lpr
```

If both lines do not start at the left border, but "stairstep" instead, see Preventing Stairstepping on Plain Text Printers.

Text files can now be printed with `lpr`. Give the filename on the command line, or pipe output directly into `lpr`.

```
% lpr textfile.txt
% ls -lh | lpr
```

11.2. Printer Connections

Printers are connected to computer systems in a variety of ways. Small desktop printers are usually connected directly to a computer's USB port. Older printers are connected to a parallel or "printer" port. Some printers are directly connected to a network, making it easy for multiple computers to share them. A few printers use a rare serial port connection.

FreeBSD can communicate with all of these types of printers.

USB

USB printers can be connected to any available USB port on the computer.

When FreeBSD detects a USB printer, two device entries are created: /dev/ulpt0 and /dev/unlpt0. Data sent to either device will be relayed to the printer. After each print job, ulpt0 resets the USB port. Resetting the port can cause problems with some printers, so the unlpt0 device is usually used instead. unlpt0 does not reset the USB port at all.

Parallel (IEEE-1284)

The parallel port device is /dev/lpt0. This device appears whether a printer is attached or not, it is not autodetected.

Vendors have largely moved away from these "legacy" ports, and many computers no longer have them. Adapters can be used to connect a parallel printer to a USB port. With such an adapter, the printer can be treated as if it were actually a USB printer. Devices called *print servers* can also be used to connect parallel printers directly to a network.

Serial (RS-232)

Serial ports are another legacy port, rarely used for printers except in certain niche applications.

Cables, connectors, and required wiring vary widely.

For serial ports built into a motherboard, the serial device name is /dev/cuau0 or /dev/cuau1. Serial `USB` adapters can also be used, and these will appear as /dev/cuaU0.

Several communication parameters must be known to communicate with a serial printer. The most important are *baud rate* or `BPS` (Bits Per Second) and *parity*. Values vary, but typical serial printers use a baud rate of 9600 and no parity.

Network

Network printers are connected directly to the local computer network.

The `DNS` hostname of the printer must be known. If the printer is assigned a dynamic address by `DHCP`, `DNS` should be dynamically updated so that the host name always has the correct `IP` address. Network printers are often given static `IP` addresses to avoid this problem.

Most network printers understand print jobs sent with the LPD protocol. A print queue name can also be specified. Some printers process data differently depending on which queue is used. For example, a `raw` queue prints the data unchanged, while the `text` queue adds carriage returns to plain text.

Many network printers can also print data sent directly to port 9100.

11.2.1. Summary

Wired network connections are usually the easiest to set up and give the fastest printing. For direct connection to the computer, `USB` is preferred for speed and simplicity. Parallel connections work but have limitations on cable length and speed. Serial connections are more difficult to configure. Cable wiring differs between models, and communication parameters like baud rate and parity bits must add to the complexity. Fortunately, serial printers are rare.

11.3. Common Page Description Languages

Data sent to a printer must be in a language that the printer can understand. These languages are called Page Description Languages, or PDLs.

`ASCII`

Plain `ASCII` text is the simplest way to send data to a printer. Characters correspond one to one with what will be printed: an `A` in the data prints an `A` on the page. Very little formatting is available. There is no way to select a font or proportional spacing. The forced simplicity of plain `ASCII` means that text can be printed straight from the computer with little or no encoding or translation. The printed output corresponds directly with what was sent.

Some inexpensive printers cannot print plain `ASCII` text. This makes them more difficult to set up, but it is usually still possible.

PostScript®

PostScript® is almost the opposite of `ASCII`. Rather than simple text, a PostScript® program is a set of instructions that draw the final document. Different fonts and graphics can be used.

However, this power comes at a price. The program that draws the page must be written. Usually this program is generated by application software, so the process is invisible to the user.

Inexpensive printers sometimes leave out PostScript® compatibility as a cost-saving measure.

`PCL` (Printer Command Language)

`PCL` is an extension of `ASCII`, adding escape sequences for formatting, font selection, and printing graphics. Many printers provide `PCL5` support. Some support the newer `PCL6` or `PCLXL`. These later versions are supersets of `PCL5` and can provide faster printing.

Host-Based

Manufacturers can reduce the cost of a printer by giving it a simple processor and very little memory. These printers are not capable of printing plain text. Instead, bitmaps of text and graphics are drawn by a driver on the host computer and then sent to the printer. These are called *host-based* printers.

Communication between the driver and a host-based printer is often through proprietary or undocumented protocols, making them functional only on the most common operating systems.

11.3.1. Converting PostScript® to Other PDLs

Many applications from the Ports Collection and FreeBSD utilities produce PostScript® output. This table shows the utilities available to convert that into other common PDLs:

Table 23. Output PDLs

Output PDL	Generated By	Notes
`PCL` or `PCL5`	print/ghostscript9-base	`-sDEVICE=ljet4` for monochrome, `-sDEVICE=cljet5` for color
`PCLXL` or `PCL6`	print/ghostscript9-base	`-sDEVICE=pxlmono` for monochrome, `-sDEVICE=pxlcolor` for color
`ESC/P2`	print/ghostscript9-base	`-sDEVICE=uniprint`
`XQX`	print/foo2zjs	

11.3.2. Summary

For the easiest printing, choose a printer that supports PostScript®. Printers that support `PCL` are the next preferred. With print/ghostscript9-base, these printers can be used as if they understood PostScript® natively. Printers that support PostScript® or `PCL` directly almost always support direct printing of plain `ASCII` text files also.

Line-based printers like typical inkjets usually do not support PostScript® or `PCL`. They often can print plain `ASCII` text files. print/ghostscript9-base supports the PDLs used by some of these printers. However, printing an entire graphic-based page on these printers is often very slow due to the large amount of data to be transferred and printed.

Host-based printers are often more difficult to set up. Some cannot be used at all because of proprietary PDLs. Avoid these printers when possible. The particular `PDL` used by various models of printers can be found at http://www.openprinting.org/printers.

11.4. Direct Printing

For occasional printing, files can be sent directly to a printer device without any setup. For example, a file called sample.txt can be sent to a `USB` printer:

```
# cp sample.txt /dev/unlpt0
```

Direct printing to network printers depends on the abilities of the printer, but most accept print jobs on port 9100, and nc(1) can be used with them. To print the same file to a printer with the `DNS` hostname of *netlaser*:

```
# nc netlaser 9100 < sample.txt
```

11.5. LPD (Line Printer Daemon)

Printing a file in the background is called *spooling*. A spooler allows the user to continue with other programs on the computer without waiting for the printer to slowly complete the print job.

FreeBSD includes a spooler called lpd(8). Print jobs are submitted with lpr(1).

11.5.1. Initial Setup

A directory for storing print jobs is created, ownership is set, and the permissions are set to prevent other users from viewing the contents of those files:

```
# mkdir -p /var/spool/lpd/lp
# chown daemon:daemon /var/spool/lpd/lp
# chmod 770 /var/spool/lpd/lp
```

Printers are defined in /etc/printcap. An entry for each printer includes details like a name, the port where it is attached, and various other settings. Create /etc/printcap with these contents:

```
lp:\                ①
    :lp=/dev/unlpt0:\   ②
    :sh:\           ③
    :mx#0:\         ④
    :sd=/var/spool/lpd/lp:\ ⑤
    :lf=/var/log/lpd-errs:  ⑥
```

① The name of this printer. lpr(1) sends print jobs to the `lp` printer unless another printer is specified with `-P`, so the default printer should be named `lp`.

② The device where the printer is connected. Replace this line with the appropriate one for the connection type shown here.

③ Suppress the printing of a header page at the start of a print job.

④ Do not limit the maximum size of a print job.

⑤ The path to the spooling directory for this printer. Each printer uses its own spooling directory.

⑥ The log file where errors on this printer will be reported.

After creating /etc/printcap, use chkprintcap(8) to test it for errors:

```
# chkprintcap
```

Fix any reported problems before continuing.

Enable lpd(8) in /etc/rc.conf:

```
lpd_enable="YES"
```

Start the service:

```
# service lpd start
```

11.5.2. Printing with lpr(1)

Documents are sent to the printer with `lpr`. A file to be printed can be named on the command line or piped into `lpr`. These two commands are equivalent, sending the contents of doc.txt to the default printer:

```
% lpr doc.txt
% cat doc.txt | lpr
```

Printers can be selected with `-P`. To print to a printer called *laser*:

```
% lpr -Plaser doc.txt
```

11.5.3. Filters

The examples shown so far have sent the contents of a text file directly to the printer. As long as the printer understands the content of those files, output will be printed correctly.

Some printers are not capable of printing plain text, and the input file might not even be plain text.

Filters allow files to be translated or processed. The typical use is to translate one type of input, like plain text, into a form that the printer can understand, like PostScript® or `PCL`. Filters can also be used to provide additional features, like adding page numbers or highlighting source code to make it easier to read.

The filters discussed here are *input filters* or *text filters*. These filters convert the incoming file into different forms. Use su(1) to become `root` before creating the files.

Filters are specified in /etc/printcap with the `if=` identifier. To use /usr/local/libexec/lf2crlf as a filter, modify /etc/printcap like this:

```
lp:\
    :lp=/dev/unlpt0:\
    :sh:\
    :mx#0:\
    :sd=/var/spool/lpd/lp:\
    :if=/usr/local/libexec/lf2crlf:\   ①
    :lf=/var/log/lpd-errs:
```

① `if=` identifies the *input filter* that will be used on incoming text.

> The backslash *line continuation* characters at the end of the lines in printcap entries reveal that an entry for a printer is really just one long line with entries delimited by colon characters. An earlier example can be rewritten as a single less-readable line:
>
> ```
> lp:lp=/dev/unlpt0:sh:mx#0:sd=/var/spool/lpd/lp:if=/usr/local/libexec/lf2crlf:lf=/var/log/lpd-errs:
> ```

Preventing Stairstepping on Plain Text Printers

Typical FreeBSD text files contain only a single line feed character at the end of each line. These lines will "stairstep" on a standard printer:

```
A printed file looks
                    like the steps of a staircase
                                             scattered by the wind
```

A filter can convert the newline characters into carriage returns and newlines. The carriage returns make the printer return to the left after each line. Create /usr/local/libexec/lf2crlf with these contents:

```
#!/bin/sh
CR=$'\r'
/usr/bin/sed -e "s/$/${CR}/g"
```

Set the permissions and make it executable:

```
# chmod 555 /usr/local/libexec/lf2crlf
```

Modify /etc/printcap to use the new filter:

```
:if=/usr/local/libexec/lf2crlf:\
```

Test the filter by printing the same plain text file. The carriage returns will cause each line to start at the left side of the page.

Fancy Plain Text on PostScript® Printers with print/enscript

GNUEnscript converts plain text files into nicely-formatted PostScript® for printing on PostScript® printers. It adds page numbers, wraps long lines, and provides numerous other features to make printed text files easier to read. Depending on the local paper size, install either print/enscript-letter or print/enscript-a4 from the Ports Collection.

Create /usr/local/libexec/enscript with these contents:

```
#!/bin/sh
/usr/local/bin/enscript -o -
```

Set the permissions and make it executable:

```
# chmod 555 /usr/local/libexec/enscript
```

Modify /etc/printcap to use the new filter:

```
:if=/usr/local/libexec/enscript:\
```

Test the filter by printing a plain text file.

Printing PostScript® to `PCL` Printers

Many programs produce PostScript® documents. However, inexpensive printers often only understand plain text or `PCL`. This filter converts PostScript® files to `PCL` before sending them to the printer.

Install the Ghostscript PostScript® interpreter, print/ghostscript9-base, from the Ports Collection.

Create /usr/local/libexec/ps2pcl with these contents:

```
#!/bin/sh
/usr/local/bin/gs -dSAFER -dNOPAUSE -dBATCH -q -sDEVICE=ljet4 -sOutputFile=- -
```

Set the permissions and make it executable:

```
# chmod 555 /usr/local/libexec/ps2pcl
```

PostScript® input sent to this script will be rendered and converted to `PCL` before being sent on to the printer.

Modify /etc/printcap to use this new input filter:

```
:if=/usr/local/libexec/ps2pcl:\
```

Test the filter by sending a small PostScript® program to it:

```
% printf "%%\!PS \n /Helvetica findfont 18 scalefont setfont \
72 432 moveto (PostScript printing successful.) show showpage \004" | lpr
```

Smart Filters

A filter that detects the type of input and automatically converts it to the correct format for the printer can be very convenient. The first two characters of a PostScript® file are usually `%!`. A filter can detect those two characters. PostScript® files can be sent on to a PostScript® printer unchanged. Text files can be converted to PostScript® with Enscript as shown earlier. Create /usr/local/libexec/psif with these contents:

```
#!/bin/sh
#
#  psif - Print PostScript or plain text on a PostScript printer
#
IFS="" read -r first_line
first_two_chars=`expr "$first_line" : '\(..\)'`

case "$first_two_chars" in
%!)
    # %! : PostScript job, print it.
    echo "$first_line" && cat && exit 0
    exit 2
    ;;
*)
    # otherwise, format with enscript
    ( echo "$first_line"; cat ) | /usr/local/bin/enscript -o - && exit 0
    exit 2
    ;;
esac
```

Set the permissions and make it executable:

```
# chmod 555 /usr/local/libexec/psif
```

Modify /etc/printcap to use this new input filter:

```
:if=/usr/local/libexec/psif:\
```

Test the filter by printing PostScript® and plain text files.

11.5.4. Multiple Queues

The entries in /etc/printcap are really definitions of *queues*. There can be more than one queue for a single printer. When combined with filters, multiple queues provide users more control over how their jobs are printed.

As an example, consider a networked PostScript® laser printer in an office. Most users want to print plain text, but a few advanced users want to be able to print PostScript® files directly. Two entries can be created for the same printer in /etc/printcap:

```
textprinter:\
    :lp=9100@officelaser:\
    :sh:\
    :mx#0:\
    :sd=/var/spool/lpd/textprinter:\
    :if=/usr/local/libexec/enscript:\
    :lf=/var/log/lpd-errs:

psprinter:\
    :lp=9100@officelaser:\
    :sh:\
    :mx#0:\
    :sd=/var/spool/lpd/psprinter:\
    :lf=/var/log/lpd-errs:
```

Documents sent to `textprinter` will be formatted by the /usr/local/libexec/enscript filter shown in an earlier example. Advanced users can print PostScript® files on `psprinter`, where no filtering is done.

This multiple queue technique can be used to provide direct access to all kinds of printer features. A printer with a duplexer could use two queues, one for ordinary single-sided printing, and one with a filter that sends the command sequence to enable double-sided printing and then sends the incoming file.

11.5.5. Monitoring and Controlling Printing

Several utilities are available to monitor print jobs and check and control printer operation.

lpq(1)

lpq(1) shows the status of a user's print jobs. Print jobs from other users are not shown.

Show the current user's pending jobs on a single printer:

```
% lpq -Plp
Rank   Owner      Job  Files                                 Total Size
1st    jsmith     0    (standard input)                      12792 bytes
```

Show the current user's pending jobs on all printers:

```
% lpq -a
lp:
Rank   Owner      Job  Files                                 Total Size
1st    jsmith     1    (standard input)                      27320 bytes

laser:
Rank   Owner      Job  Files                                 Total Size
1st    jsmith     287  (standard input)                      22443 bytes
```

lprm(1)

lprm(1) is used to remove print jobs. Normal users are only allowed to remove their own jobs. `root` can remove any or all jobs.

Remove all pending jobs from a printer:

```
# lprm -Plp -
dfA002smithy dequeued
cfA002smithy dequeued
dfA003smithy dequeued
cfA003smithy dequeued
dfA004smithy dequeued
cfA004smithy dequeued
```

Remove a single job from a printer. lpq(1) is used to find the job number.

```
% lpq
Rank   Owner      Job  Files                                 Total Size
1st    jsmith     5    (standard input)                      12188 bytes

% lprm -Plp 5
dfA005smithy dequeued
cfA005smithy dequeued
```

lpc(8)

lpc(8) is used to check and modify printer status. `lpc` is followed by a command and an optional printer name. `all` can be used instead of a specific printer name, and the command will be applied to all printers. Normal users can view status with lpc(8). Only `root` can use commands which modify printer status.

Show the status of all printers:

```
% lpc status all
lp:
    queuing is enabled
    printing is enabled
    1 entry in spool area
    printer idle
laser:
    queuing is enabled
    printing is enabled
    1 entry in spool area
    waiting for laser to come up
```

Prevent a printer from accepting new jobs, then begin accepting new jobs again:

```
# lpc disable lp
lp:
    queuing disabled
# lpc enable lp
lp:
    queuing enabled
```

Stop printing, but continue to accept new jobs. Then begin printing again:

```
# lpc stop lp
lp:
    printing disabled
# lpc start lp
lp:
    printing enabled
    daemon started
```

Restart a printer after some error condition:

```
# lpc restart lp
lp:
    no daemon to abort
    printing enabled
    daemon restarted
```

Turn the print queue off and disable printing, with a message to explain the problem to users:

```
# lpc down lp Repair parts will arrive on Monday
lp:
    printer and queuing disabled
    status message is now: Repair parts will arrive on Monday
```

Re-enable a printer that is down:

```
# lpc up lp
lp:
    printing enabled
    daemon started
```

See lpc(8) for more commands and options.

11.5.6. Shared Printers

Printers are often shared by multiple users in businesses and schools. Additional features are provided to make sharing printers more convenient.

Aliases

The printer name is set in the first line of the entry in /etc/printcap. Additional names, or *aliases*, can be added after that name. Aliases are separated from the name and each other by vertical bars:

```
lp|repairsprinter|salesprinter:\
```

Aliases can be used in place of the printer name. For example, users in the Sales department print to their printer with

```
% lpr -Psalesprinter sales-report.txt
```

Users in the Repairs department print to *their* printer with

```
% lpr -Prepairsprinter repairs-report.txt
```

All of the documents print on that single printer. When the Sales department grows enough to need their own printer, the alias can be removed from the shared printer entry and used as the name of a new printer. Users in both departments continue to use the same commands, but the Sales documents are sent to the new printer.

Header Pages

It can be difficult for users to locate their documents in the stack of pages produced by a busy shared printer. *Header pages* were created to solve this problem. A header page with the user name and document name is printed before each print job. These pages are also sometimes called *banner* or *separator* pages.

Enabling header pages differs depending on whether the printer is connected directly to the computer with a `USB`, parallel, or serial cable, or is connected remotely over a network.

Header pages on directly-connected printers are enabled by removing the `:sh:\` (Suppress Header) line from the entry in /etc/printcap. These header pages only use line feed characters for new lines. Some printers will need the /usr/share/examples/printing/hpif filter to prevent stairstepped text. The filter configures `PCL` printers to print both carriage returns and line feeds when a line feed is received.

Header pages for network printers must be configured on the printer itself. Header page entries in /etc/printcap are ignored. Settings are usually available from the printer front panel or a configuration web page accessible with a web browser.

11.5.7. References

Example files: /usr/share/examples/printing/.

The *4.3BSD Line Printer Spooler Manual*, /usr/share/doc/smm/07.lpd/paper.ascii.gz.

Manual pages: printcap(5), lpd(8), lpr(1), lpc(8), lprm(1), lpq(1).

11.6. Other Printing Systems

Several other printing systems are available in addition to the built-in lpd(8). These systems offer support for other protocols or additional features.

11.6.1. CUPS (Common UNIX® Printing System)

CUPS is a popular printing system available on many operating systems. Using CUPS on FreeBSD is documented in a separate article: CUPS

11.6.2. HPLIP

Hewlett Packard provides a printing system that supports many of their inkjet and laser printers. The port is print/hplip. The main web page is at https://developers.hp.com/hp-linux-imaging-and-printing. The port handles all the installation details on FreeBSD. Configuration information is shown at https://developers.hp.com/hp-linux-imaging-and-printing/install.

11.6.3. LPRng

LPRng was developed as an enhanced alternative to lpd(8). The port is sysutils/LPRng. For details and documentation, see https://lprng.sourceforge.net/.

Chapter 12. Linux Binary Compatibility

12.1. Synopsis

FreeBSD provides **optional** binary compatibility with Linux®, commonly referred to as Linuxulator, allowing users to install and run unmodified Linux binaries. It is available for the x86 (both 32 and 64 bit) and AArch64 architectures. Some Linux-specific operating system features are not yet supported; this mostly happens with functionality specific to hardware or related to system management, such as cgroups or namespaces.

Before reading this chapter:

- Know how to install additional third-party software.

Read this chapter to learn:

- How to enable Linux binary compatibility on a FreeBSD system.
- How to install additional Linux shared libraries.
- How to install Linux applications on a FreeBSD system.
- The implementation details of Linux compatibility in FreeBSD.

12.2. Configuring Linux Binary Compatibility

By default, linux(4) binary compatibility is not enabled.

To enable the Linux ABI at boot time, execute the following command:

```
# sysrc linux_enable="YES"
```

Once enabled, it can be started without rebooting by executing the following command:

```
# service linux start
```

This is enough for statically linked Linux binaries to work.

The Linux service will load necessary kernel modules and mount filesystems expected by Linux applications under /compat/linux. They can be started in the same way native FreeBSD binaries can; they behave almost exactly like native processes and can be traced and debugged the usual way.

The current content of /compat/linux can be checked executing the following command:

```
# ls -l /compat/linux/
```

The output should be similar to the following:

```
total 1
dr-xr-xr-x  13 root  wheel   512 Apr 11 19:12 dev
dr-xr-xr-x   1 root  wheel     0 Apr 11 21:03 proc
dr-xr-xr-x   1 root  wheel     0 Apr 11 21:03 sys
```

12.3. Linux userlands

Linux software requires more than just an ABI to work. In order to run Linux software a Linux userland must be installed first.

If all that is wanted is to run some software already included in the Ports tree, it can be installed via package manager and pkg(8) will automatically setup the required Linux userland.

For example, to install Sublime Text 4, along with all the Linux libraries it depends on, run this command:

```
# pkg install linux-sublime-text4
```

12.3.1. Rocky Linux Base System from FreeBSD Packages

To install the Rocky Linux 9 userland execute the following command:

```
# pkg install linux_base-rl9
```

emulators/linux_base-rl9 will place the base system derived from Rocky Linux 9 into /compat/linux.

After installing the package, the contents of /compat/linux can be verified by running the following command to check that the Rocky Linux userland has been installed:

```
# ls -l /compat/linux/
```

The output should be similar to the following:

```
total 36
drwxr-xr-x   2 root wheel  512 Oct  9 17:28 afs
lrwxr-xr-x   1 root wheel    7 May 16  2022 bin -> usr/bin
drwxr-xr-x   3 root wheel  512 Oct  9 17:28 dev
drwxr-xr-x  24 root wheel 1536 Oct  9 17:28 etc
lrwxr-xr-x   1 root wheel    7 May 16  2022 lib -> usr/lib
lrwxr-xr-x   1 root wheel    9 May 16  2022 lib64 -> usr/lib64
drwxr-xr-x   2 root wheel  512 Oct  9 17:28 opt
```

```
drwxr-xr-x   2 root wheel  512 Oct  9 17:28 proc
lrwxr-xr-x   1 root wheel    8 Oct  1 03:11 run -> /var/run
lrwxr-xr-x   1 root wheel    8 May 16  2022 sbin -> usr/sbin
drwxr-xr-x   2 root wheel  512 Oct  9 17:28 srv
drwxr-xr-x   2 root wheel  512 Oct  9 17:28 sys
drwxr-xr-x   8 root wheel  512 Oct  9 17:28 usr
drwxr-xr-x  16 root wheel  512 Oct  9 17:28 var
```

12.3.2. CentOS Base System from FreeBSD Packages

> ⚠ emulators/linux_base-c7 has been deprecated following the deprecation of the upstream project. This means emulators/linux_base-c7 will not receive security updates. The use of the Rocky Linux Base System is recommended unless 32-bit compatibility is needed.

To install the CentOS userland execute the following command:

```
# pkg install linux_base-c7
```

emulators/linux_base-c7 will place the base system derived from CentOS 7 into /compat/linux.

After installing the package, the contents of /compat/linux can be verified by running the following command to check that the CentOS userland has been installed:

```
# ls -l /compat/linux/
```

The output should be similar to the following:

```
total 30
lrwxr-xr-x   1 root  wheel    7 Apr 11  2018 bin -> usr/bin
drwxr-xr-x  13 root  wheel  512 Apr 11 21:10 dev
drwxr-xr-x  25 root  wheel   64 Apr 11 21:10 etc
lrwxr-xr-x   1 root  wheel    7 Apr 11  2018 lib -> usr/lib
lrwxr-xr-x   1 root  wheel    9 Apr 11  2018 lib64 -> usr/lib64
drwxr-xr-x   2 root  wheel    2 Apr 11 21:10 opt
dr-xr-xr-x   1 root  wheel    0 Apr 11 21:25 proc
lrwxr-xr-x   1 root  wheel    8 Feb 18 02:10 run -> /var/run
lrwxr-xr-x   1 root  wheel    8 Apr 11  2018 sbin -> usr/sbin
drwxr-xr-x   2 root  wheel    2 Apr 11 21:10 srv
dr-xr-xr-x   1 root  wheel    0 Apr 11 21:25 sys
drwxr-xr-x   8 root  wheel    9 Apr 11 21:10 usr
drwxr-xr-x  16 root  wheel   17 Apr 11 21:10 var
```

12.3.3. Debian / Ubuntu Base System with debootstrap

An alternative way of providing Linux shared libraries is by using sysutils/debootstrap. This has the advantage of providing a full Debian or Ubuntu distribution.

To install debootstrap execute the following command:

```
# pkg install debootstrap
```

debootstrap(8) needs linux(4) ABI enabled. Once enabled, execute the following command to install Ubuntu or Debian in /compat/ubuntu:

```
# debootstrap jammy /compat/ubuntu
```

While it is technically possible to install into /compat/linux instead, it's discouraged due to possible clashes with CentOS-based packages. Instead, derive the directory name from the distribution or version name, e.g., /compat/ubuntu.

The output should be similar to the following:

```
I: Retrieving InRelease
I: Checking Release signature
I: Valid Release signature (key id F6ECB3762474EDA9D21B7022871920D1991BC93C)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Checking component main on http://archive.ubuntu.com/ubuntu...
[...]
I: Configuring console-setup...
I: Configuring kbd...
I: Configuring ubuntu-minimal...
I: Configuring libc-bin...
I: Configuring ca-certificates...
I: Base system installed successfully.
```

Then set up mounts in /etc/fstab.

If the contents of the home directory should be shared and to be able to run X11 applications, /home and /tmp should be mounted in the linux compat area using nullfs(5) for loopback.

The following example can be added to /etc/fstab:

```
# Device        Mountpoint              FStype          Options
Dump    Pass#
```

```
devfs           /compat/ubuntu/dev      devfs           rw,late
0       0
tmpfs           /compat/ubuntu/dev/shm  tmpfs
rw,late,size=1g,mode=1777    0       0
fdescfs         /compat/ubuntu/dev/fd   fdescfs
rw,late,linrdlnk             0       0
linprocfs       /compat/ubuntu/proc     linprocfs       rw,late
0       0
linsysfs        /compat/ubuntu/sys      linsysfs        rw,late
0       0
/tmp            /compat/ubuntu/tmp      nullfs          rw,late
0       0
/home           /compat/ubuntu/home     nullfs          rw,late
0       0
```

Then execute mount(8):

```
# mount -al
```

To access the system using chroot(8) execute the following command:

```
# chroot /compat/ubuntu /bin/bash
```

Then uname(1) can be executed to check the Linux environment:

```
# uname -s -r -m
```

The output should be similar to the following:

```
Linux 5.15.0 x86_64
```

Once inside the chroot, the system behaves as in a normal Ubuntu installation. While systemd doesn't work, the service(8) command works as usual.

To add the package repositories missing from defaults edit the file /compat/ubuntu/etc/apt/sources.list.

For amd64 the following example can be used:

```
deb http://archive.ubuntu.com/ubuntu jammy main universe restricted
multiverse
deb http://security.ubuntu.com/ubuntu/ jammy-security universe
multiverse restricted main
deb http://archive.ubuntu.com/ubuntu jammy-backports universe
multiverse restricted main
```

```
deb http://archive.ubuntu.com/ubuntu jammy-updates universe multiverse
restricted main
```

For arm64 this other example can be used:

```
deb http://ports.ubuntu.com/ubuntu-ports bionic main universe
restricted multiverse
```

12.4. Advanced Topics

A list of all Linux-related sysctl(8) knobs can be found in linux(4).

Some applications require specific filesystems to be mounted.

This is normally handled by the /etc/rc.d/linux script but can be disabled at boot executing the following command:

```
sysrc linux_mounts_enable="NO"
```

Filesystems mounted by the rc script will not work for Linux processes inside chroots or jails; if needed, configure them in /etc/fstab:

```
devfs      /compat/linux/dev      devfs      rw,late                   0  0
tmpfs      /compat/linux/dev/shm  tmpfs      rw,late,size=1g,mode=1777 0  0
fdescfs    /compat/linux/dev/fd   fdescfs    rw,late,linrdlnk          0  0
linprocfs  /compat/linux/proc     linprocfs  rw,late                   0  0
linsysfs   /compat/linux/sys      linsysfs   rw,late                   0  0
```

Since the Linux binary compatibility layer has gained support for running both 32- and 64-bit Linux binaries, it is no longer possible to link the emulation functionality statically into a custom kernel.

12.4.1. Installing Additional Libraries Manually

For base system subdirectories created with debootstrap(8), use the instructions above instead.

If a Linux application complains about missing shared libraries after configuring Linux binary compatibility, determine which shared libraries the Linux binary needs and install them manually.

From a Linux system using the same CPU architecture, `ldd` can be used to determine which shared libraries the application needs.

For example, to check which shared libraries `linuxdoom` needs, run this command from a Linux system that has Doom installed:

```
% ldd linuxdoom
```

The output should be similar to the following:

```
libXt.so.3 (DLL Jump 3.1) => /usr/X11/lib/libXt.so.3.1.0
libX11.so.3 (DLL Jump 3.1) => /usr/X11/lib/libX11.so.3.1.0
libc.so.4 (DLL Jump 4.5pl26) => /lib/libc.so.4.6.29
```

Then, copy all the files in the last column of the output from the Linux system into /compat/linux on the FreeBSD system. Once copied, create symbolic links to the names in the first column.

This example will result in the following files on the FreeBSD system:

```
/compat/linux/usr/X11/lib/libXt.so.3.1.0
/compat/linux/usr/X11/lib/libXt.so.3 -> libXt.so.3.1.0
/compat/linux/usr/X11/lib/libX11.so.3.1.0
/compat/linux/usr/X11/lib/libX11.so.3 -> libX11.so.3.1.0
/compat/linux/lib/libc.so.4.6.29
/compat/linux/lib/libc.so.4 -> libc.so.4.6.29
```

If a Linux shared library already exists with a matching major revision number to the first column of the `ldd` output, it does not need to be copied to the file named in the last column, as the existing library should work. It is advisable to copy the shared library if it is a newer version, though. The old one can be removed, as long as the symbolic link points to the new one.

For example, these libraries already exist on the FreeBSD system:

```
/compat/linux/lib/libc.so.4.6.27
/compat/linux/lib/libc.so.4 -> libc.so.4.6.27
```

and `ldd` indicates that a binary requires a later version:

```
libc.so.4 (DLL Jump 4.5pl26) -> libc.so.4.6.29
```

Since the existing library is only one or two versions out of date in the last digit, the program should still work with the slightly older version. However, it is safe to replace the existing libc.so with the newer version:

```
/compat/linux/lib/libc.so.4.6.29
/compat/linux/lib/libc.so.4 -> libc.so.4.6.29
```

Generally, one will need to look for the shared libraries that Linux binaries depend on only the first few times that a Linux program is installed on FreeBSD. After a while, there will be a sufficient set of Linux shared libraries on the system to be able to run newly installed Linux binaries without any extra work.

12.4.2. Branding Linux ELF Binaries

The FreeBSD kernel uses several methods to determine if the binary to be executed is a Linux one: it checks the brand in the ELF file header, looks for known ELF interpreter paths and checks ELF notes; finally, by default, unbranded ELF executables are assumed to be Linux anyway.

Should all those methods fail, an attempt to execute the binary might result in error message:

```
% ./my-linux-elf-binary
```

The output should be similar to the following:

```
ELF binary type not known
Abort
```

To help the FreeBSD kernel distinguish between a FreeBSD ELF binary and a Linux binary, use brandelf(1):

```
% brandelf -t Linux my-linux-elf-binary
```

12.4.3. Installing a Linux RPM Based Application

To install a Linux RPM-based application, first install the archivers/rpm4 package or port. Once installed, `root` can use this command to install a .rpm:

```
# cd /compat/linux
# rpm2cpio < /path/to/linux.archive.rpm | cpio -id
```

If necessary, `brandelf` the installed ELF binaries. Note that this will prevent a clean uninstall.

12.4.4. Configuring the Hostname Resolver

If DNS does not work or this error appears:

```
resolv+: "bind" is an invalid keyword resolv+:
"hosts" is an invalid keyword
```

configure /compat/linux/etc/host.conf as follows:

```
order hosts, bind
```

```
multi on
```

This specifies that /etc/hosts is searched first and DNS is searched second. When /compat/linux/etc/host.conf does not exist, Linux applications use /etc/host.conf in the host system but they complain since that file does not exist in FreeBSD. Remove `bind` if a name server is not configured using /etc/resolv.conf.

12.4.5. Miscellaneous

More information on how binary compatibility works with Linux® can be found in the article Linux emulation in FreeBSD.

Chapter 13. WINE

13.1. Synopsis

WINE, which stands for Wine Is Not an Emulator, is technically a software translation layer. It allows installing and running software written for Windows® on FreeBSD (and other) systems.

It operates by intercepting system calls, or requests from the software to the operating system, and translating them from Windows® calls to calls that FreeBSD understands. It will also translate any responses as needed into what the Windows® software is expecting. So in some ways, it *emulates* a Windows® environment, in that it provides many of the resources Windows® applications are expecting.

However, it is not an emulator in the traditional sense. Many of these solutions operate by constructing an entirely separate computer using software processes in place of hardware. Virtualization (such as that provided by the emulators/qemu port) operates in this way. One of the benefits of this approach is the ability to install a full version of the OS in question to the emulator. It means that the environment will not look any different to applications than a real machine, and chances are good that everything will work on it. The downside to this approach is the fact that software acting as hardware is inherently slower than actual hardware. The computer built in software (called the *guest*) requires resources from the real machine (the *host*), and holds on to those resources for as long as it is running.

The WINE Project, on the other hand, is much lighter on system's resources. It will translate system calls on the fly, so while it is difficult to be as fast as a real Windows® computer, it can come very close. On the other hand, WINE is trying to keep up with a moving target in terms of all the different system calls and other functionality it needs to support. As a result there may be applications that do not work as expected on WINE, will not work at all, or will not even install to begin with.

At the end of the day, WINE provides another option to try to get a particular Windows® software program running on FreeBSD. It can always serve as the first option which, if successful, offers a good experience without unnecessarily depleting the host FreeBSD system's resources.

This chapter will describe:

- How to install WINE on a FreeBSD system.
- How WINE operates, and how it is different from other alternatives like virtualization.
- How to fine-tune WINE to the specific needs of some applications.
- How to install GUI helpers for WINE.
- Common tips and solutions for using WINE on FreeBSD.
- Considerations for WINE on FreeBSD in terms of the multi-user environment.

Before reading this chapter, it will be useful to:

- Understand the basics of UNIX® and FreeBSD.

- Know how to install FreeBSD.
- Know how to set up a network connection.
- Know how to install additional third-party software.

13.2. WINE Overview & Concepts

WINE is a complex system, so before running it on a FreeBSD system it is worth gaining an understanding of what it is and how it works.

13.2.1. What is WINE?

As mentioned in the Synopsis for this chapter, WINE is a compatibility layer that allows Windows® applications to run on other operating systems. In theory, it means these programs should run on systems like FreeBSD, macOS, and Android.

When WINE runs a Windows® executable, two things occur:

- Firstly, WINE implements an environment that mimics that of various versions of Windows®. For example, if an application requests access to a resource such as RAM, WINE has a memory interface that looks and acts (as far as the application is concerned) like Windows®.
- Then, once that application makes use of that interface, WINE takes the incoming request for space in memory and translates it to something compatible with the host system. In the same way when the application retrieves that data, WINE facilitates fetching it from the host system and passing it back to the Windows® application.

13.2.2. WINE and the FreeBSD System

Installing WINE on a FreeBSD system will entail a few different components:

- FreeBSD applications for tasks such as running the Windows® executables, configuring the WINE sub-system, or compiling programs with WINE support.
- A large number of libraries that implement the core functions of Windows® (for example /lib/wine/api-ms-core-memory-l1-1-1.dll.so, which is part of the aforementioned memory interface).
- A number of Windows® executables, which are (or mimic) common utilities (such as /lib/wine/notepad.exe.so, which provides the standard Windows® text editor).
- Additional Windows® assets, in particular fonts (like the Tahoma font, which is stored in share/wine/fonts/tahoma.ttf in the install root).

13.2.3. Graphical Versus Text Mode/Terminal Programs in WINE

As an operating system where terminal utilities are "first-class citizens," it is natural to assume that WINE will contain extensive support for text-mode program. However, the majority of applications for Windows®, especially the most popular ones, are designed with a graphical user interface (GUI) in mind. Therefore, WINE's utilities are designed by default to launch graphical programs.

However, there are three methods available to run these so-called Console User Interface (CUI) programs:

- The *Bare Streams* approach will display the output directly to standard output.
- The *wineconsole* utility can be used with either the *user* or *curses* backend to utilize some of the enhancements the WINE system provides for CUI applications.

These approaches are described in greater detail on the WINE Wiki.

13.2.4. WINE Derivative Projects

WINE itself is a mature open source project, so it is little surprise it is used as the foundation of more complex solutions.

Commercial WINE Implementations

A number of companies have taken WINE and made it a core of their own, proprietary products (WINE's LGPL license permits this). Two of the most famous of these are as follows:

- Codeweavers CrossOver

This solution provides a simplified "one-click" installation of WINE, which contains additional enhancements and optimizations (although the company contributes many of these back upstream to the WINE project). One area of focus for Codeweavers is to make the most popular applications install and run smoothly.

While the company once produced a native FreeBSD version of their CrossOver solution, it appears to have long been abandoned. While some resources (such as a dedicated forum) are still present, they also have seen no activity for some time.

- Steam Proton

Gaming company Steam also uses WINE to enable Windows® games to install and run on other systems. it is primary target is Linux-based systems, though some support exists for macOS as well.

While Steam does not offer a native FreeBSD client, there are several options for using the Linux® client using FreeBSD's Linux Compatibility Layer.

WINE Companion Programs

In addition to proprietary offerings, other projects have released applications designed to work in tandem with the standard, open source version of WINE. The goals for these can range from making installation easier to offering easy ways to get popular software installed.

These solutions are covered in greater detail in the later section on GUI frontends, and include the following:

- winetricks
- Mizutamari

13.2.5. Alternatives to WINE

For FreeBSD users, some alternatives to using WINE are as follows:

- Dual-Booting: A straightforward option is to run desired Windows® applications natively on that OS. This of course means exiting FreeBSD in order to boot Windows®, so this method is not feasible if access to programs in both systems is required simultaneously.
- Virtual Machines: Virtual Machines (VMs), as mentioned earlier in this chapter, are software processes that emulate full sets of hardware, on which additional operating systems (including Windows®) can be installed and run. Modern tools make VMs easy to create and manage, but this method comes at a cost. A good portion of the host systems resources must be allocated to each VM, and those resources cannot be reclaimed by the host as long as the VM is running. A few examples of VM managers include the open source solutions qemu, bhyve, and VirtualBox. See the chapter on Virtualization for more detail.
- Remote Access: Like many other UNIX®-like systems, FreeBSD can run a variety of applications enabling users to remotely access Windows® computers and use their programs or data. In addition to clients such as xrdp that connect to the standard Windows® Remote Desktop Protocol, other open source standards such as vnc can also be used (provided a compatible server is present on the other side).

13.3. Installing WINE on FreeBSD

WINE can be installed via the pkg tool, or by compiling the port(s).

13.3.1. WINE Prerequistes

Before installing WINE itself, it is useful to have the following pre-requisites installed.

- A GUI

Most Windows® programs are expecting to have a graphical user interface available. If WINE is installed without one present, its dependencies will include the Wayland compositor, and so a GUI will be installed along with WINE. But it is useful to have the GUI of choice installed, configured, and working correctly before installing WINE.

- wine-gecko

The Windows® operating system has for some time had a default web browser pre-installed: Internet Explorer. As a result, some applications work under the assumption that there will always be something capable of displaying web pages. In order to provide this functionality, the WINE layer includes a web browser component using the Mozilla project's Gecko engine. When WINE is first launched it will offer to download and install this, and there are reasons users might want it do so (these will be covered in a later chapter). But they can also install it prior to installing WINE, or alongside the install of WINE proper.

Install this package with the following:

```
# pkg install wine-gecko
```

Alternately, compile the port with the following:

```
# cd /usr/ports/emulator/wine-gecko
# make install
```

- wine-mono

This port installs the MONO framework, an open source implementation of Microsoft's .NET. Including this with the WINE installation will make it that much more likely that any applications written in .NET will install and run on the system.

To install the package:

```
# pkg install wine-mono
```

To compile from the ports collection:

```
# cd /usr/ports/emulator/wine-mono
# make install
```

13.3.2. Installing WINE via FreeBSD Package Repositories

With the pre-requisites in place, install WINE via package with the following command:

```
# pkg install wine
```

Alternately compile the WINE sub-system from source with the following:

```
# cd /usr/ports/emulator/wine
# make install
```

13.3.3. Concerns of 32- Versus 64-Bit in WINE Installations

Like most software, Windows® applications made the upgrade from the older 32-bit architecture to 64 bits. And most recent software is written for 64-bit operating systems, although modern OSes can sometimes continue to run older 32-bit programs as well. FreeBSD is no different, having had support for 64-bit since the 5.x series.

However, using old software no longer supported by default is a common use for emulators, and users commonly turn to WINE to play games and use other programs that do not run properly on modern hardware. Fortunately, FreeBSD can support all three scenarios:

- On modern, 64-bit machine and want to run 64-bit Windows® software, simply install the ports mentioned in the above sections. The ports system will automatically install the 64-bit version.
- Alternately, users might have an older 32-bit machine that they do not want to run with its original, now non-supported software. They can install the 32-bit (i386) version of FreeBSD, then install the ports in the above sections.

13.4. Running a First WINE Program on FreeBSD

Now that WINE is installed, the next step is to try it out by running a simple program. An easy way to do this is to download a self-contained application, i.e., one can simply unpack and run without any complex installation process.

So-called "portable" versions of applications are good choices for this test, as are programs that run with only a single executable file.

13.4.1. Running a Program from the Command Line

There are two different methods to launch a Windows program from the terminal. The first, and most straightforward is to navigate to the directory containing the program's executable (.EXE) and issue the following:

```
% wine program.exe
```

For applications that take command-line arguments, add them after the executable as usual:

```
% wine program2.exe -file file.txt
```

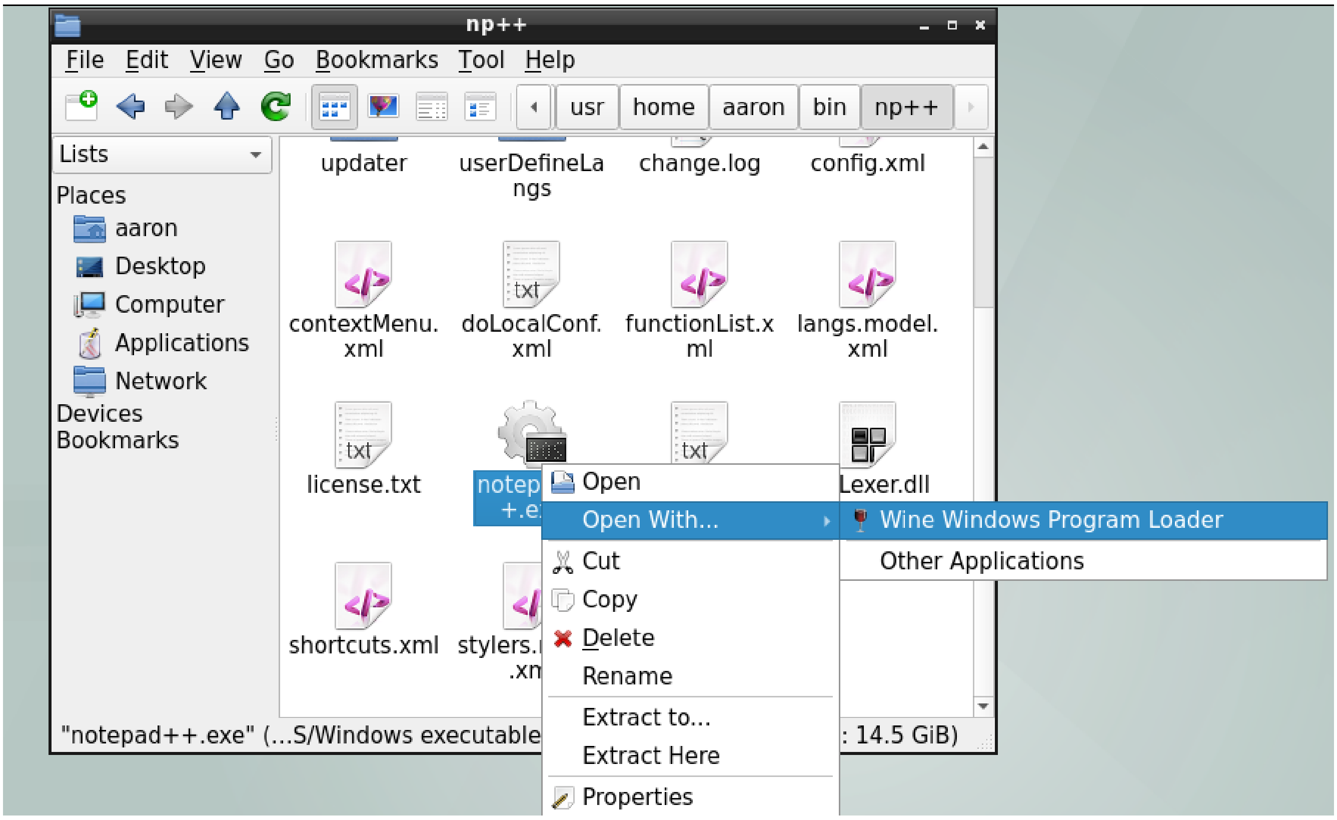
Alternately, supply the full path to the executable to use it in a script, for example:

```
% wine /home/user/bin/program.exe
```

13.4.2. Running a Program from a GUI

After installation graphical shells should be updated with new associations for Windows executable (.EXE) files. It will now be possible to browse the system using a file manager, and launch the Windows application in the same way as other files and programs (either a single- or double-click, depending on the desktop's settings).

On most desktops, check to make sure this association is correct by right-clicking on the file, and looking for an entry in the context menu to open the file. One of the options (hopefully the default one) will be with the **Wine Windows Program Loader**, as shown in the below screenshot:


np++
File Edit View Go Bookmarks Tool Help
usr home aaron bin np++
Lists
Places
aaron
Desktop
Computer
Applications
Network
Devices
Bookmarks
updater
userDefineLangs
change.log
config.xml
contextMenu.xml
doLocalConf.xml
functionList.xml
langs.model.xml
license.txt
notepad++.exe
Lexer.dll
shortcuts.xml
stylers.xml
Open
Open With...
Wine Windows Program Loader
Other Applications
Cut
Copy
Delete
Rename
Extract to...
Extract Here
Properties
"notepad++.exe" (...S/Windows executable
: 14.5 GiB)

In the event the program does not run as expected, try launching it from the command line and review any messages displayed in the terminal to troubleshoot.

In the event WINE is not the default application for .EXE files after install, check the MIME associate for this extension in the current desktop environment, graphical shell, or file manager.

13.5. Configuring WINE Installation

With an understanding of what WINE is and how it works at a high level, the next step to effectively using it on FreeBSD is becoming familiar with its configuration. The following sections will describe the key concept of the *WINE prefix*, and illustrate how it is used to control the behavior of applications run through WINE.

13.5.1. WINE Prefixes

A WINE *prefix* is a directory, usually located beneath the default location of $HOME/.wine though it can be located elsewhere. The prefix is a set of configurations and support files used by the wine to configure and run the Windows® environment a given application needs. By default, a brand new WINE installation will create the following structure when first launched by a user:

- .update-timestamp: contains the last modified date of file /usr/share/wine/wine.inf. It is used by WINE to determine if a prefix is out of date, and automatically update it if needed.
- dosdevices/: contains information on mappings of Windows® resources to resources on the host FreeBSD system. For example, after a new WINE installation, this should contain at least two entries which enable access to the FreeBSD filesystem using Windows®-style drive letters:
  - c:@: A link to drive_c described below.
  - z:@: A link to the root directory of the system.

- drive_c/: emulates the main (i.e., C:) drive of a Windows® system. It contains a directory structure and associated files mirroring that of standard Windows® systems. A fresh WINE prefix will contain Windows® 10 directories such as *Users* and *Windows* that holds the OS itself. Furthermore, applications installed within a prefix will be located in either *Program Files* or *Program Files (x86)*, depending on their architecture.
- system.reg: This Registry file contains information on the Windows® installation, which in the case of WINE is the environment in drive_c.
- user.reg: This Registry file contains the current user's personal configurations, made either by various software or through the use of the Registry Editor.
- userdef.reg: This Registry file is a default set of configurations for newly-created users.

13.5.2. Creating and Using WINE Prefixes

While WINE will create a default prefix in the user's $HOME/.wine/, it is possible to set up multiple prefixes. There are a few reasons to do this:

- The most common reason is to emulate different versions of Windows®, according to the compatibility needs of the software in question.
- In addition, it is common to encounter software that does not work correctly in the default environment, and requires special configuration. it is useful to isolate these in their own, custom prefixes, so the changes do not impact other applications.
- Similarly, copying the default or "main" prefix into a separate "testing" one in order to evaluate an application's compatibility can reduce the chance of corruption.

Creating a prefix from the terminal requires the following command:

```
% WINEPREFIX="/home/username/.wine-new" winecfg
```

This will run the `winecfg` program, which can be used to configure wine prefixes (more on this in a later section). But by providing a directory path value for the `WINEPREFIX` environment variable, a new prefix is created at that location if one does not already exist.

Supplying the same variable to the wine program will similarly cause the selected program to be run with the specified prefix:

```
% WINEPREFIX="/home/username/.wine-new" wine program.exe
```

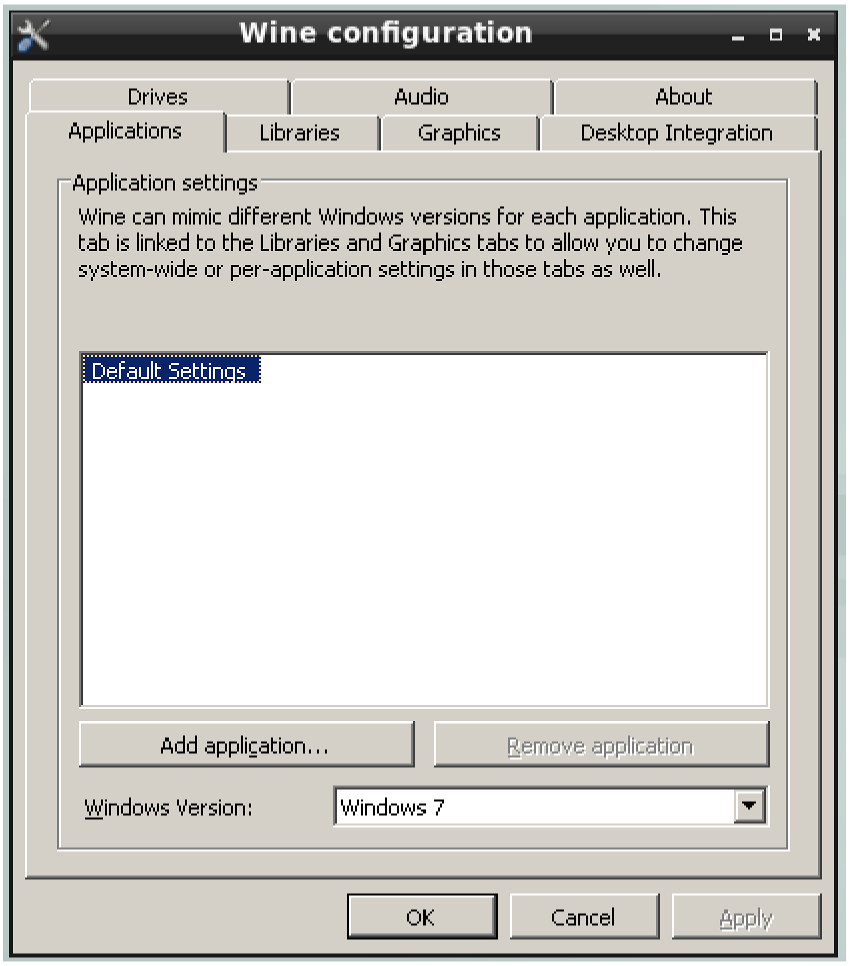
13.5.3. Configuring WINE Prefixes with winecfg

As described above WINE includes a tool called `winecfg` to configure prefixes from within a GUI. It contains a variety of functions, which are detailed in the sections below. When `winecfg` is run from within a prefix, or provided the location of a prefix within the `WINEPREFIX` variable, it enables the configuration of the selected prefix as described in the below sections.

Selections made on the *Applications* tab will affect the scope of changes made in the *Libraries* and

Graphics tabs, which will be limited to the application selected. See the section on Using Winecfg in the WINE Wiki for more details.

Applications


Wine configuration
Drives
Audio
About
Applications
Libraries
Graphics
Desktop Integration
Application settings
Default Settings
Add application...
Remove application
OK
Cancel
Apply

Wine can mimic different Windows versions for each application. This tab is linked to the Libraries and Graphics tabs to allow you to change system-wide or per-application settings in those tabs as well.

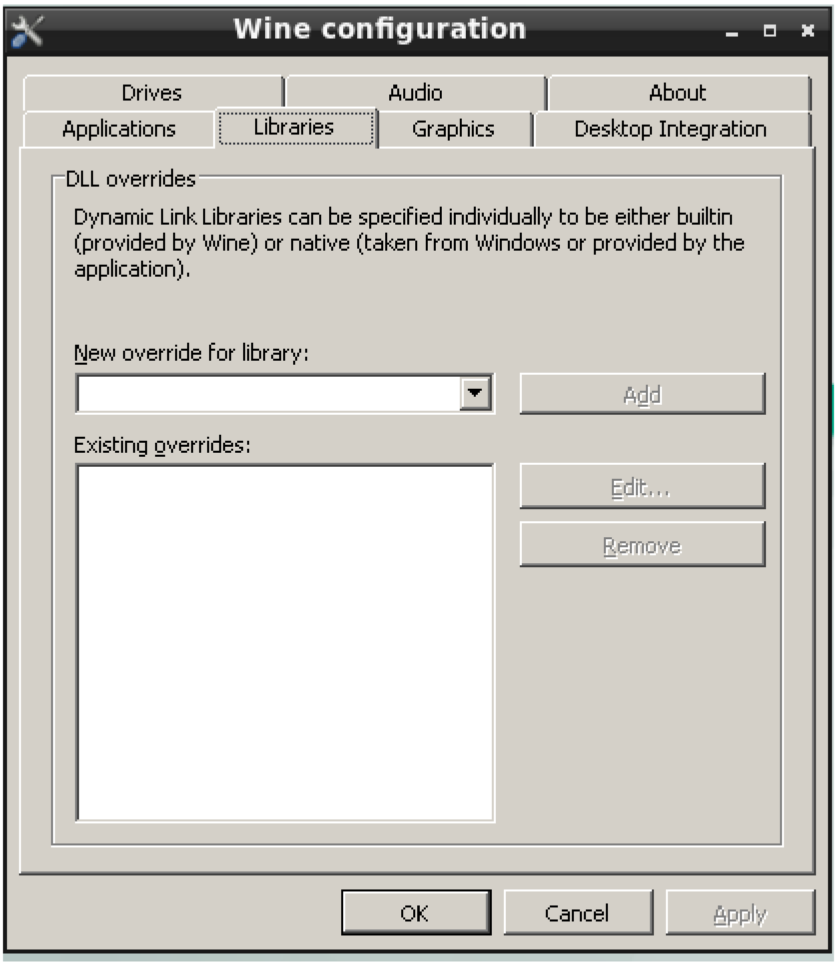
Windows Version: Windows 7

The *Applications* contains controls enabling the association of programs with a particular version of Windows®. On first start-up the *Application settings* section will contain a single entry: *Default Settings*. This corresponds to all the default configurations of the prefix, which (as the disabled *Remove application* button implies) cannot be deleted.

But additional applications can be added with the following process:

1. Click the *Add application* button.
2. Use the provided dialog to select the desired program's executable.
3. Select the version of Windows® to be used with the selected program.

Libraries


Wine configuration
Drives
Audio
About
Applications
Libraries
Graphics
Desktop Integration
DLL overrides
New override for library:
Add
Existing overrides:
Edit...
Remove
OK
Cancel
Apply

Dynamic Link Libraries can be specified individually to be either builtin (provided by Wine) or native (taken from Windows or provided by the application).

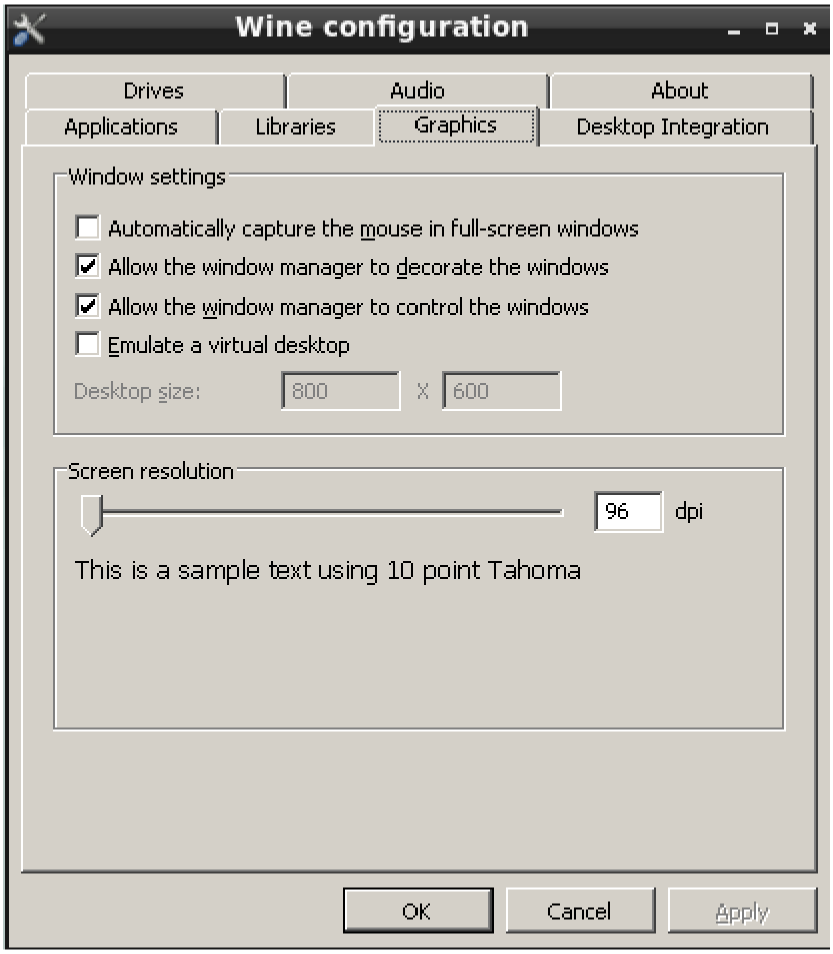
WINE provides a set of open source library files as part of its distribution that provide the same functions as their Windows® counterparts. However, as noted earlier in this chapter, the WINE project is always trying to keep pace with new updates to these libraries. As a result, the versions that ship with WINE may be missing functionality that the latest Windows® programs are expecting.

However, `winecfg` makes it possible specify overrides for the built-in libraries, particularly there is a version of Windows® available on the same machine as the host FreeBSD installation. For each library to be overridden, do the following:

1. Open the *New override for library* drop-down and select the library to be replaced.
2. Click the *Add* button.
3. The new override will appear in the *Existing overrides* list, notice the *native, builtin* designation in parentheses.
4. Click to select the library.
5. Click the *Edit* button.
6. Use the provided dialog to select a corresponding library to be used in place of the built-in one.

Be sure to select a file that is truly the corresponding version of the built-in one, otherwise there may be unexpected behavior.

Graphics


Wine configuration
Drives
Audio
About
Applications
Libraries
Graphics
Desktop Integration
OK
Cancel
Apply

Window settings

☐ Automatically capture the mouse in full-screen windows

☑ Allow the window manager to decorate the windows

☑ Allow the window manager to control the windows

☐ Emulate a virtual desktop

Desktop size: 800 X 600

Screen resolution

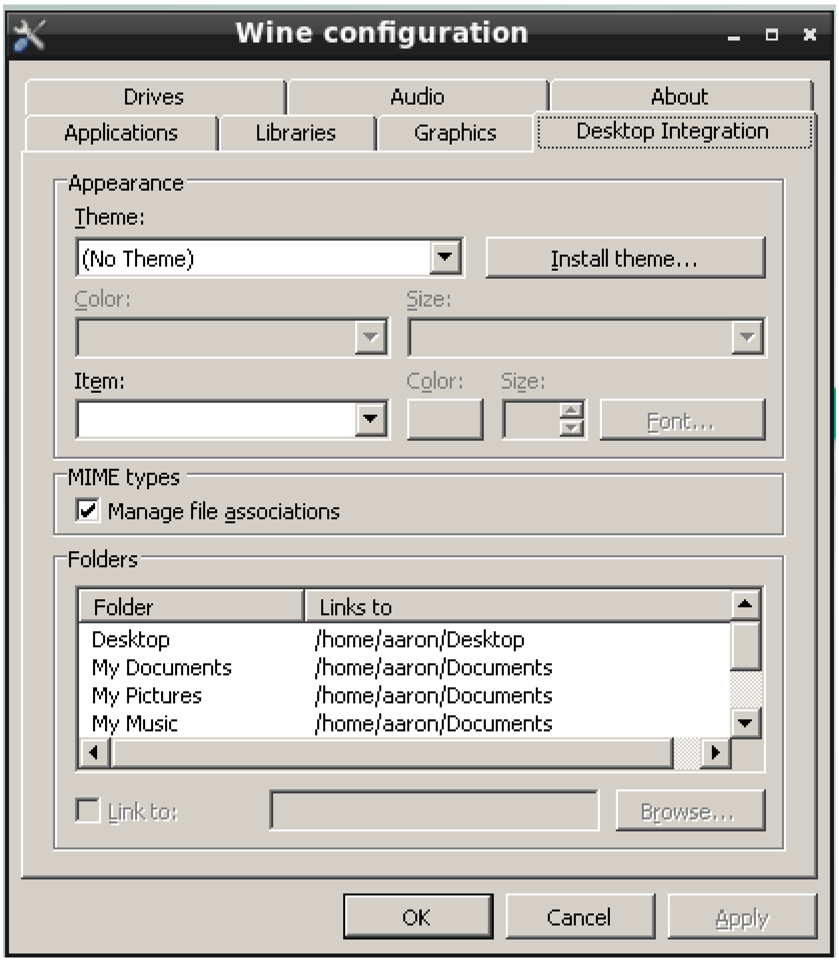
96 dpi

This is a sample text using 10 point Tahoma

The *Graphics* tab provides some options to make the windows of programs run via WINE operate smoothly with FreeBSD:

- Automatic mouse capture when windows are full-screen.
- Allowing the FreeBSD window manager to decorate the windows, such as their title bars, for programs running via WINE.
- Allowing the window manager to control windows for programs running via WINE, such as running resizing functions on them.
- Create an emulated virtual desktop, within which all WINE programs will run. If this item is selected, the size of the virtual desktop can be specified using the *Desktop size* input boxes.
- Setting the screen resolution for programs running via WINE.

Desktop Integration


Wine configuration
Drives
Audio
About
Applications
Libraries
Graphics
Desktop Integration
Appearance
Color:
Size:
Item:
Color:
Size:
Install theme...
Font...
MIME types
Folders
Link to:
Browse...
OK
Cancel
Apply

Theme: (No Theme)

☑ Manage file associations

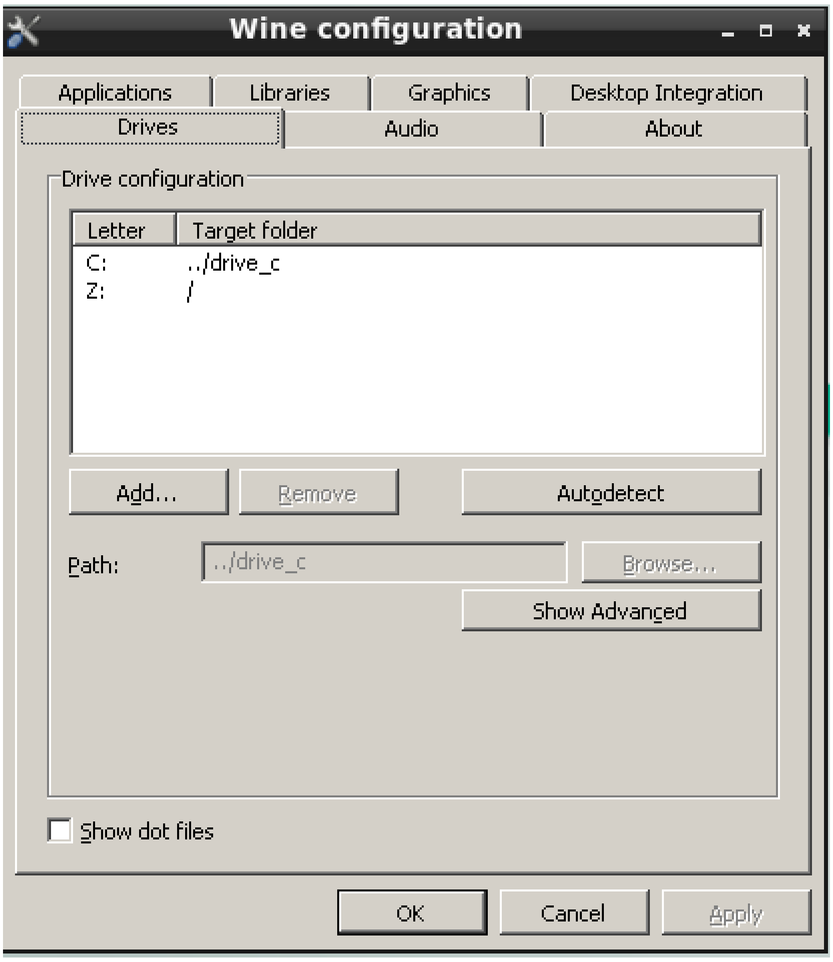
Folder	Links to
Desktop	/home/aaron/Desktop
My Documents	/home/aaron/Documents
My Pictures	/home/aaron/Documents
My Music	/home/aaron/Documents

☐ Link to:

This tab allows configuration of the following items:

- The theme and related visual settings to be used for programs running via WINE.
- Whether the WINE sub-system should manage MIME types (used to determine which application opens a particular file type) internally.
- Mappings of directories in the host FreeBSD system to useful folders within the Windows® environment. To change an existing association, select the desired item and click *Browse*, then use the provided dialog to select a directory.

Drives


Wine configuration
Applications
Libraries
Graphics
Desktop Integration
Drives
Audio
About
Drive configuration
Add...
Remove
Autodetect
Browse...
Show Advanced
☐ Show dot files
OK
Cancel
Apply

Letter	Target folder
C:	../drive_c
Z:	/

Path: ../drive_c

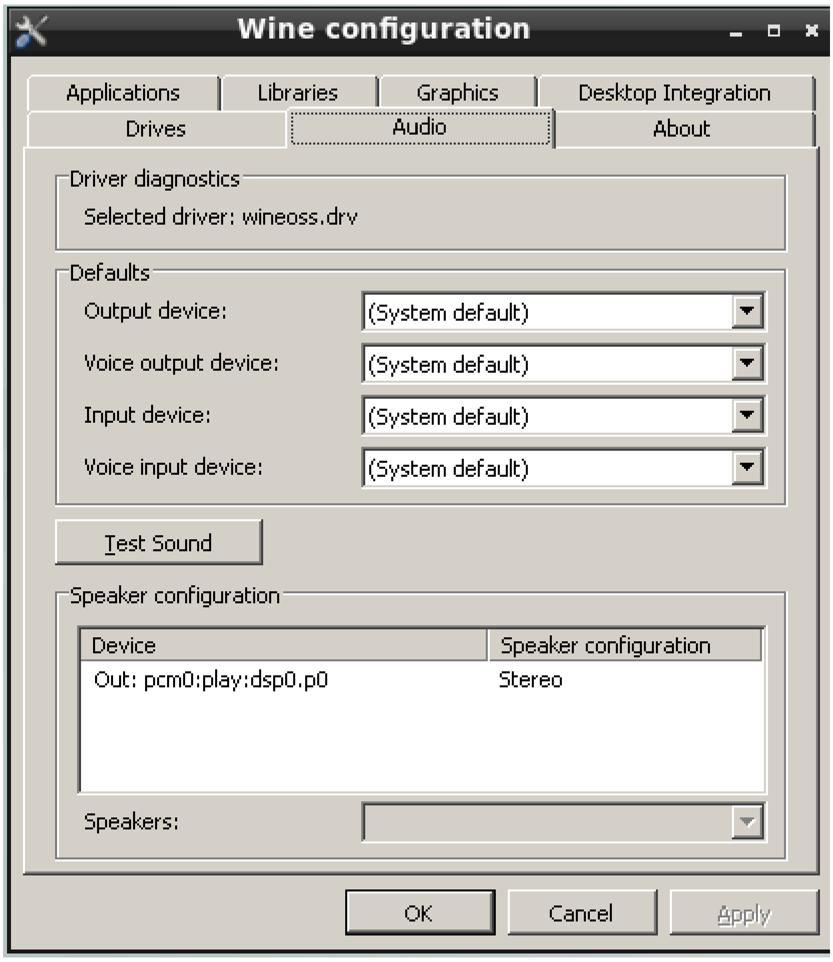
The *Drives* tab allows linking of directories in the host FreeBSD system to drive letters in the Windows® environment. The default values in this tab should look familiar, as they are displaying the contents of dosdevices/ in the current WINE prefix. Changes made via this dialog will reflect in dosdevices, and properly-formatted links created in that directory will display in this tab.

To create a new entry, such as for a CD-ROM (mounted at /mnt/cdrom), take the following steps:

1. Click the _Add _ button.
2. In the provided dialog, choose a free drive letter.
3. Click *OK*.
4. Fill in the *Path* input box by either typing the path to the resource, or click _Browse _ and use the provided dialog to select it.

By default WINE will autodetect the type of resource linked, but this can be manually overridden. See the section in the WINE Wiki for more detail on advanced options.

Audio


Wine configuration
Applications
Libraries
Graphics
Desktop Integration
Drives
Audio
About
Driver diagnostics
Defaults
Test Sound
Speaker configuration
OK
Cancel
Apply

Selected driver: wineoss.drv

Output device: (System default)

Voice output device: (System default)

Input device: (System default)

Voice input device: (System default)

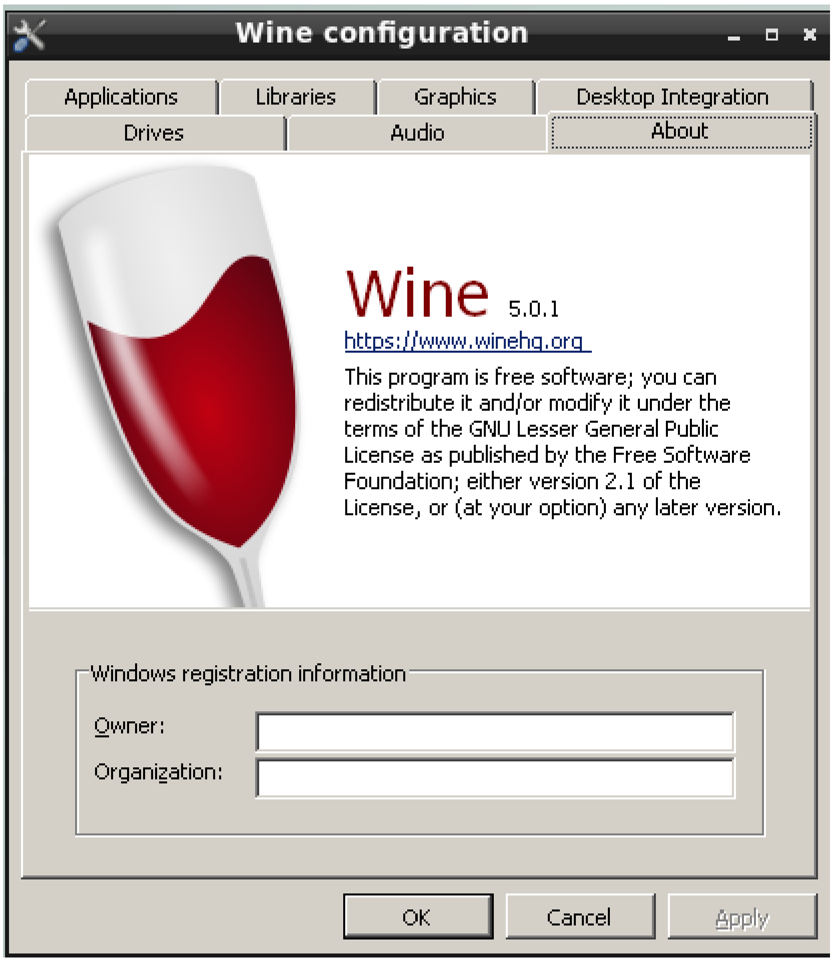
Device	Speaker configuration
Out: pcm0:play:dsp0.p0	Stereo

Speakers:

This tab contains some configurable options for routing sound from Windows® programs to the native FreeBSD sound system, including:

- Driver selection
- Default device selection
- Sound test

About


Wine configuration
Applications
Libraries
Graphics
Desktop Integration
Drives
Audio
About
Windows registration information
Owner:
Organization:
OK
Cancel
Apply

Wine 5.0.1

https://www.winehq.org

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

The final tab contains information on the WINE project, including a link to the website. It also allows entry of (entirely optional) user information, although this is not sent anywhere as it is in other operating systems.

13.6. WINE Management GUIs

While the base install of WINE comes with a GUI configuration tool in `winecfg`, it is main purpose is just that: strictly configuring an existing WINE prefix. There are, however, more advanced applications that will assist in the initial installation of applications as well as optimizing their WINE environments. The below sections include a selection of the most popular.

13.6.1. Winetricks

The `winetricks` tool is a cross-platform, general purpose helper program for WINE. It is not developed by the WINE project proper, but rather maintained on GitHub by a group of contributors. It contains some automated "recipes" for getting common applications to work on WINE, both by optimizing the settings as well as acquiring some DLL libraries automatically.

Installing winetricks

To install `winetricks` on a FreeBSD using binary packages, use the following commands (note `winetricks` requires either the i386-wine or i386-wine-devel package, and is therefore not installed automatically with other dependencies):

```
# pkg install i386-wine winetricks
```

To compile it from source, issue the following in the terminal:

```
# cd /usr/ports/emulators/i386-wine
# make install
# cd /usr/ports/emulators/winetricks
# make install
```

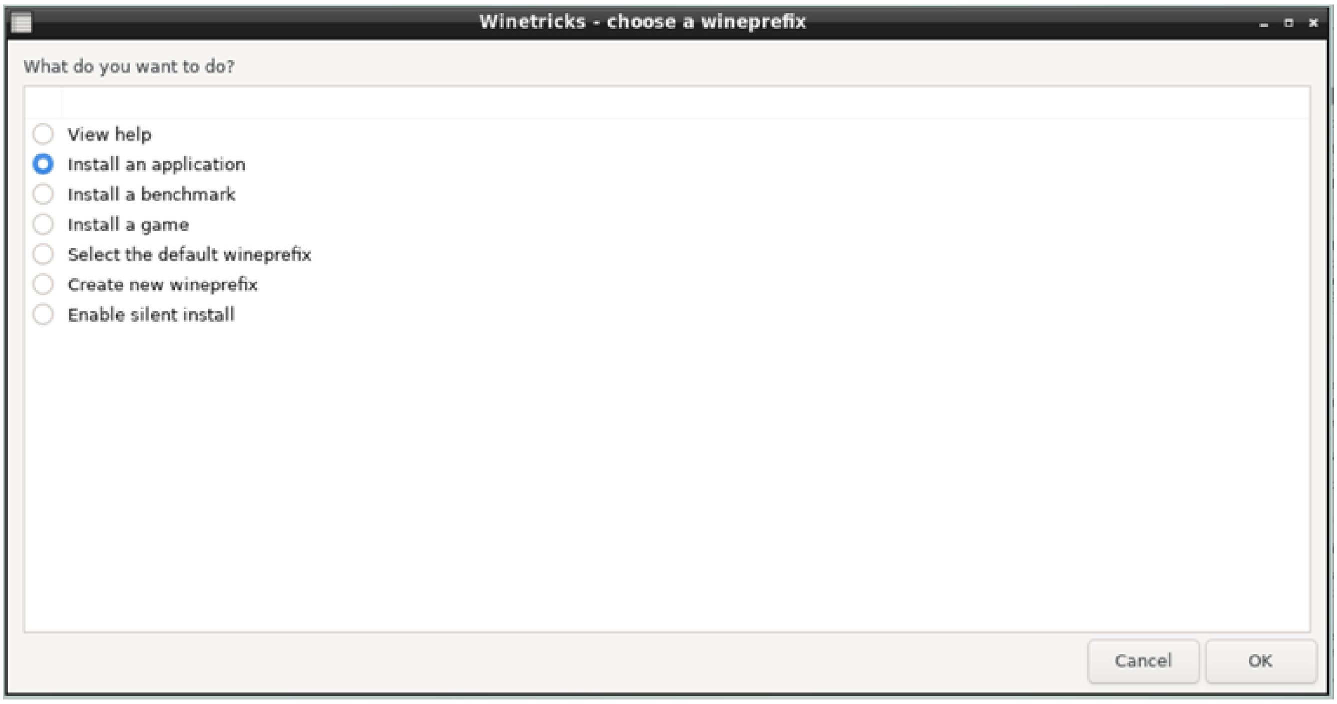
If a manual installation is required, refer to the Github account for instructions.

Using winetricks

Run `winetricks` with the following command:

```
% winetricks
```

Note: this should be in a 32-bit prefix to run `winetricks`. Launching `winetricks` displays a window with a number of choices, as follows:

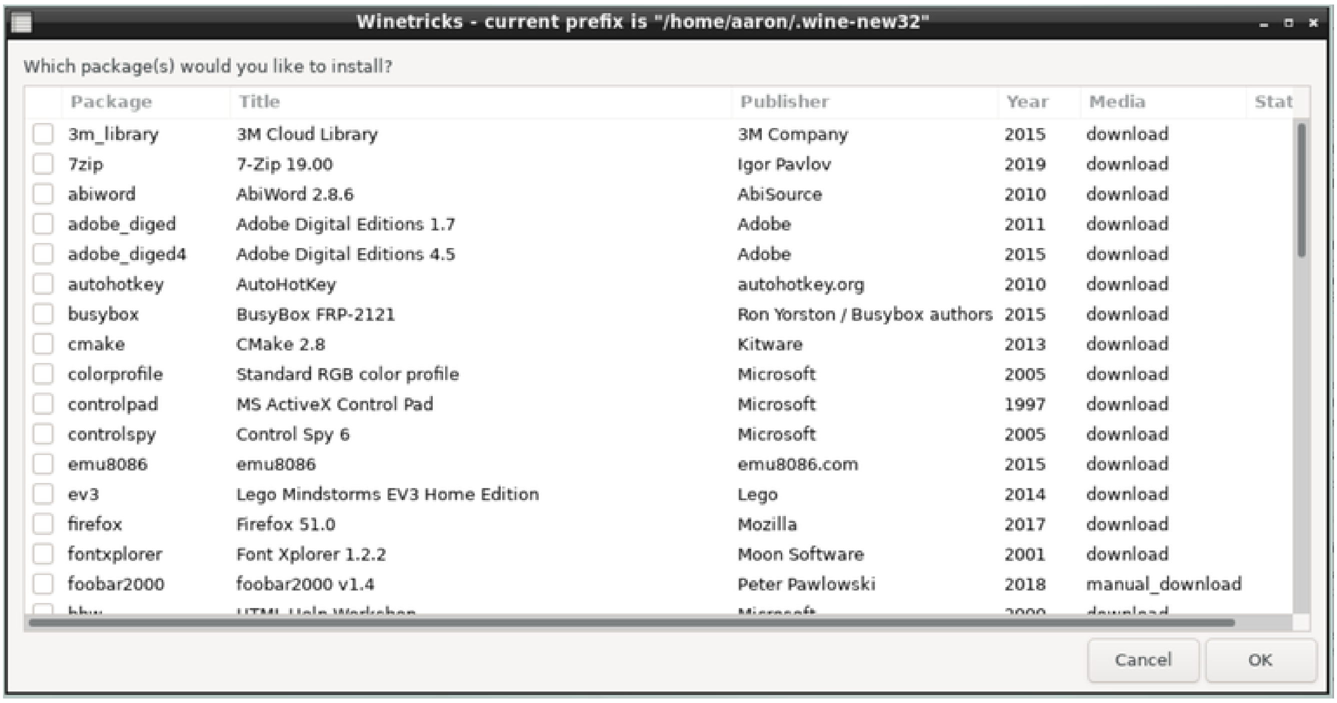


Winetricks - choose a wineprefix

What do you want to do?

○ View help
● Install an application
○ Install a benchmark
○ Install a game
○ Select the default wineprefix
○ Create new wineprefix
○ Enable silent install

Selecting either *Install an application*, *Install a benchmark*, or *Install a game* shows a list with supported options, such as the one below for applications:

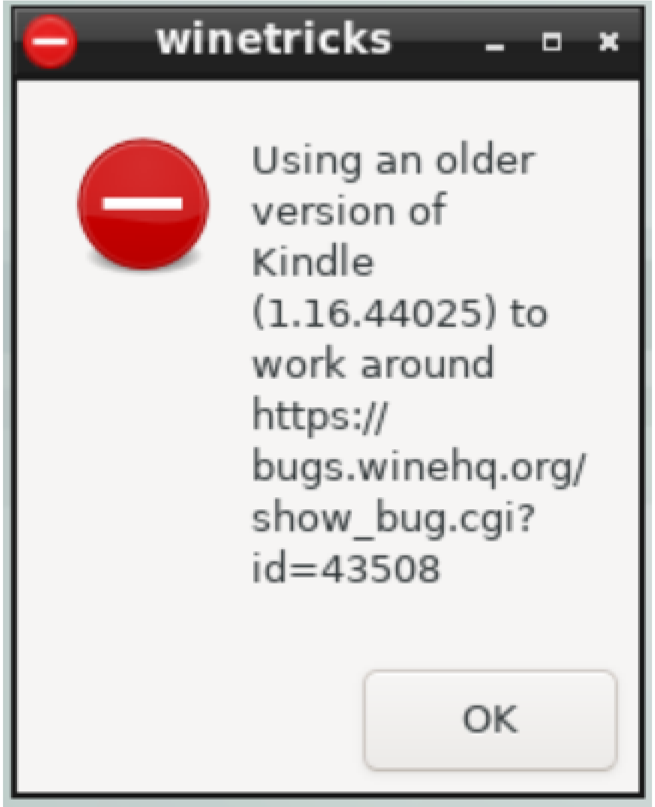


Winetricks - current prefix is "/home/aaron/.wine-new32"

Which package(s) would you like to install?

	Package	Title	Publisher	Year	Media	Stat
☐	3m_library	3M Cloud Library	3M Company	2015	download	
☐	7zip	7-Zip 19.00	Igor Pavlov	2019	download	
☐	abiword	AbiWord 2.8.6	AbiSource	2010	download	
☐	adobe_diged	Adobe Digital Editions 1.7	Adobe	2011	download	
☐	adobe_diged4	Adobe Digital Editions 4.5	Adobe	2015	download	
☐	autohotkey	AutoHotKey	autohotkey.org	2010	download	
☐	busybox	BusyBox FRP-2121	Ron Yorston / Busybox authors	2015	download	
☐	cmake	CMake 2.8	Kitware	2013	download	
☐	colorprofile	Standard RGB color profile	Microsoft	2005	download	
☐	controlpad	MS ActiveX Control Pad	Microsoft	1997	download	
☐	controlspy	Control Spy 6	Microsoft	2005	download	
☐	emu8086	emu8086	emu8086.com	2015	download	
☐	ev3	Lego Mindstorms EV3 Home Edition	Lego	2014	download	
☐	firefox	Firefox 51.0	Mozilla	2017	download	
☐	fontxplorer	Font Xplorer 1.2.2	Moon Software	2001	download	
☐	foobar2000	foobar2000 v1.4	Peter Pawlowski	2018	manual_download	

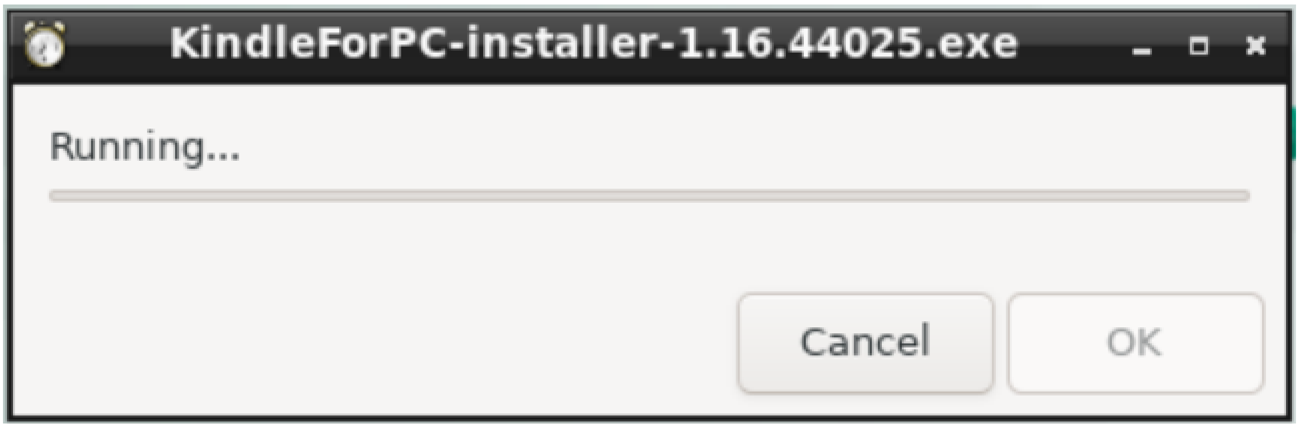
Selecting one or more items and clicking *OK* will start their installation process(es). Initially, some messages that appear to be errors may show up, but they're actually informational alerts as `winetricks` configures the WINE environment to get around known issues for the application:



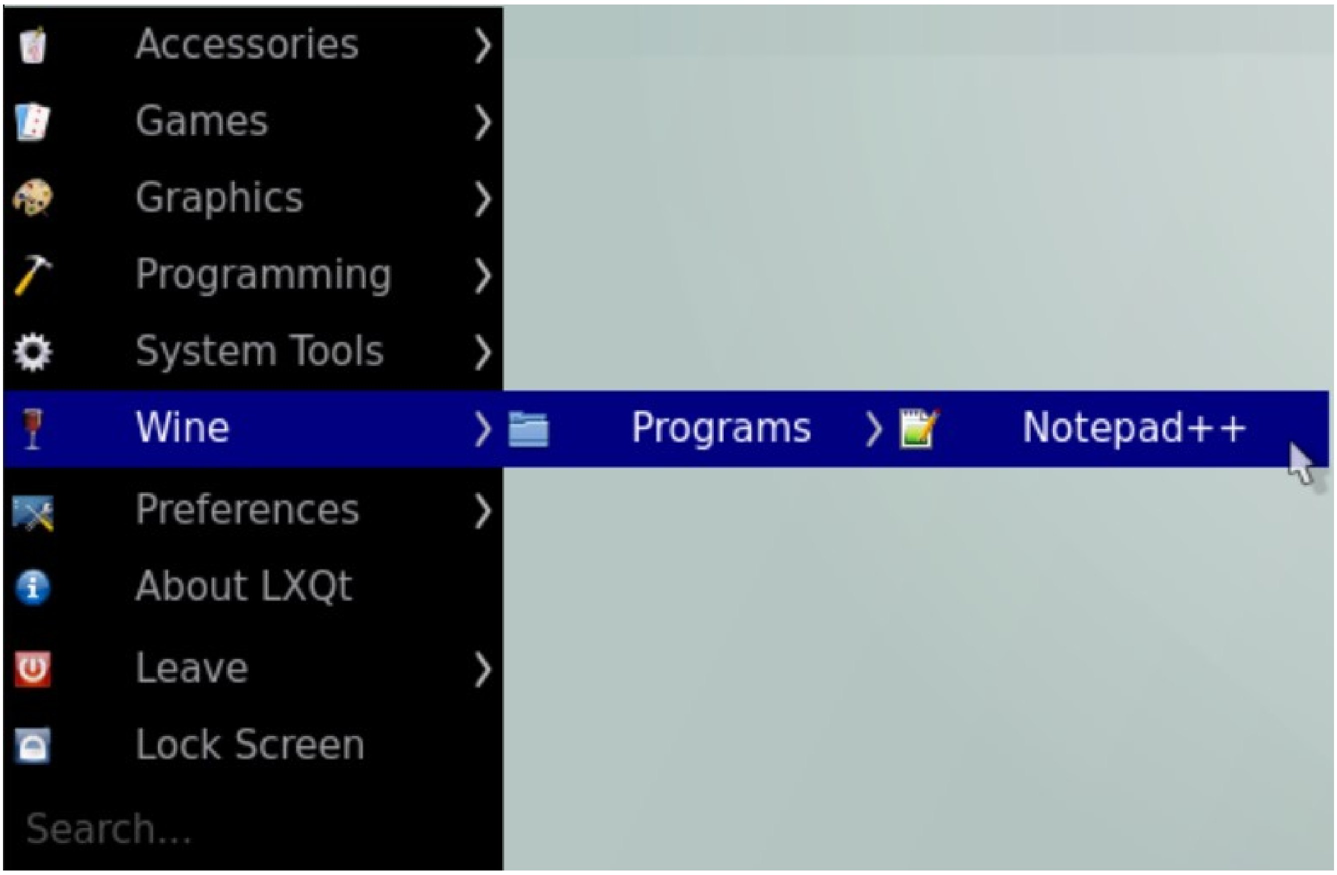
winetricks

Using an older version of Kindle (1.16.44025) to work around https://bugs.winehq.org/show_bug.cgi?id=43508

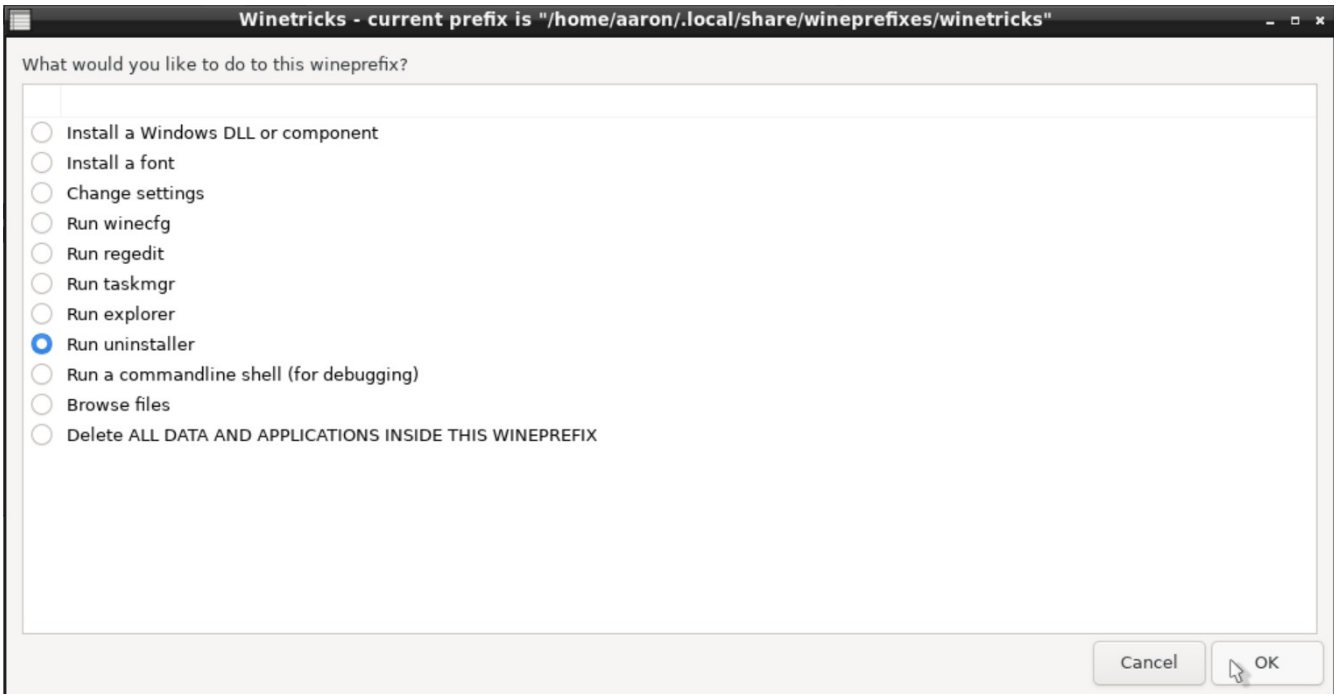
Once these are circumvented, the actual installer for the application will be run:


KindleForPC-installer-1.16.44025.exe
Running...
Cancel
OK

Once the installation completes, the new Windows application should be available from the desktop environment's standard menu (shown in the screenshot below for the LXQT desktop environment):


Accessories
Games
Graphics
Programming
System Tools
Wine
Programs
Notepad++
Preferences
About LXQt
Leave
Lock Screen
Search...

In order to remove the application, run `winetricks` again, and select *Run an uninstaller*.

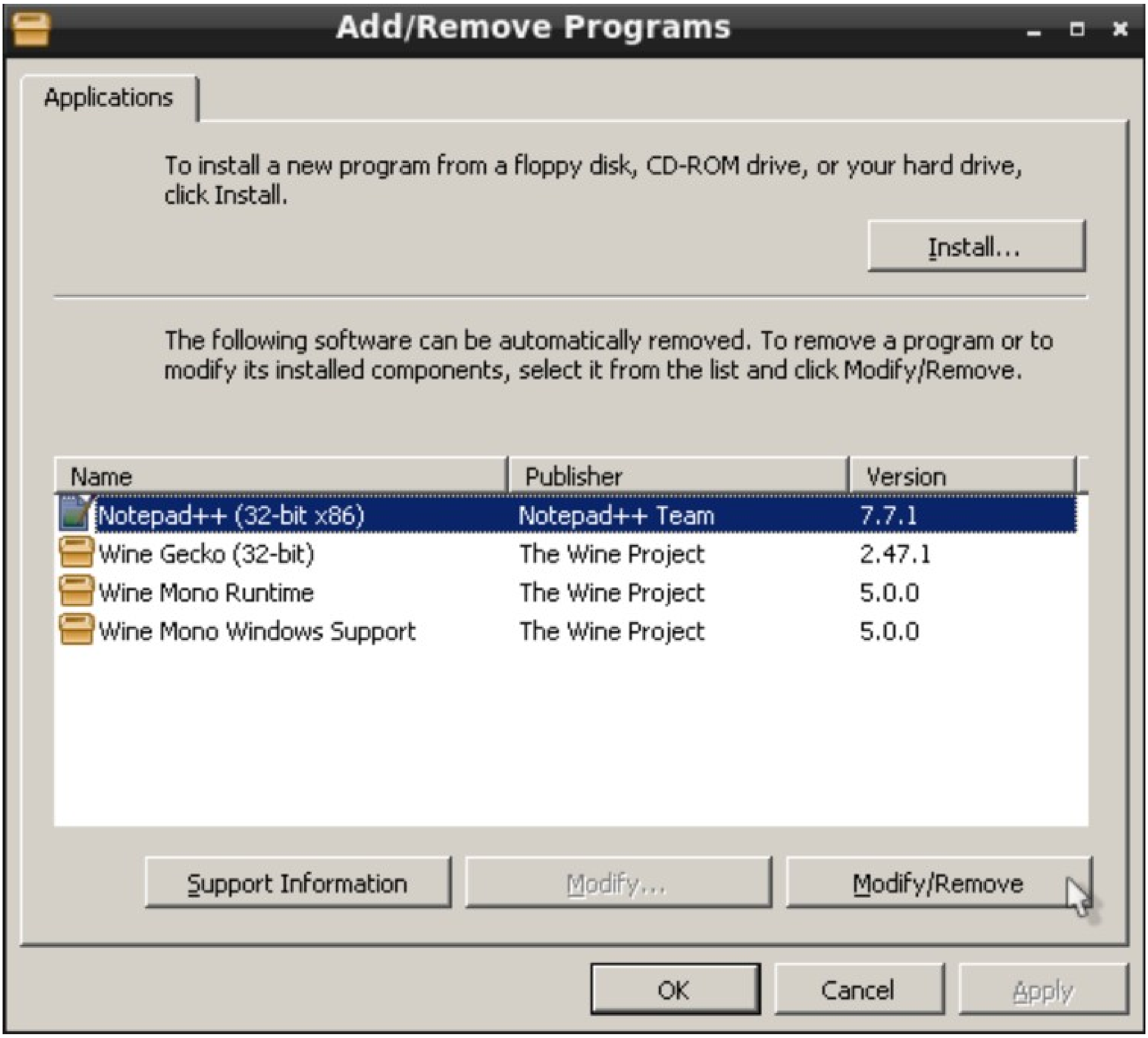


Winetricks - current prefix is "/home/aaron/.local/share/wineprefixes/winetricks"

What would you like to do to this wineprefix?

- ○ Install a Windows DLL or component
- ○ Install a font
- ○ Change settings
- ○ Run winecfg
- ○ Run regedit
- ○ Run taskmgr
- ○ Run explorer
- ● Run uninstaller
- ○ Run a commandline shell (for debugging)
- ○ Browse files
- ○ Delete ALL DATA AND APPLICATIONS INSIDE THIS WINEPREFIX

A Windows®-style dialog will appear with a list of installed programs and components. Select the application to be removed, then click the *Modify/Remove* button.

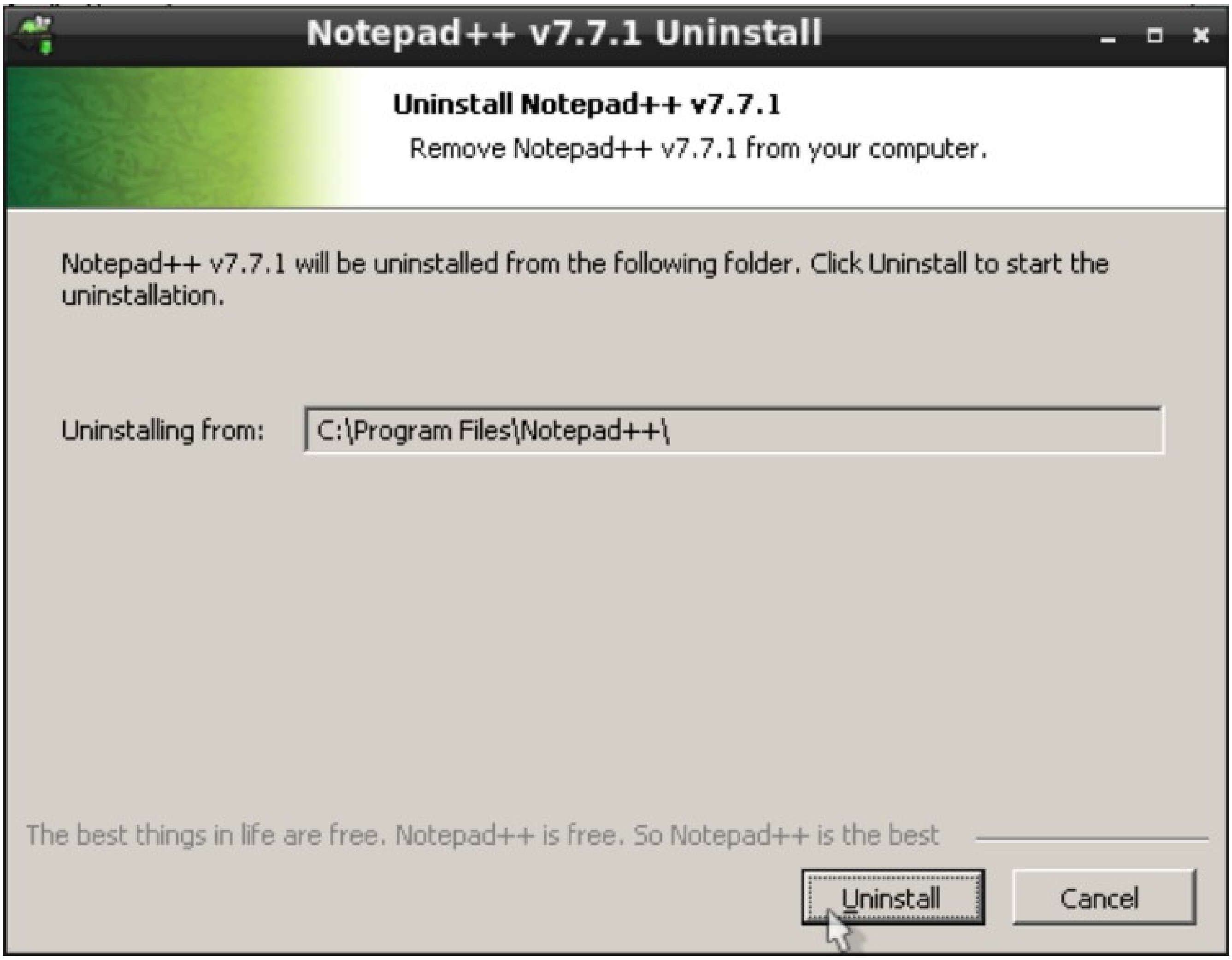

Add/Remove Programs
Applications
Install...
Support Information
Modify...
Modify/Remove
OK
Cancel
Apply

To install a new program from a floppy disk, CD-ROM drive, or your hard drive, click Install.

The following software can be automatically removed. To remove a program or to modify its installed components, select it from the list and click Modify/Remove.

Name	Publisher	Version
Notepad++ (32-bit x86)	Notepad++ Team	7.7.1
Wine Gecko (32-bit)	The Wine Project	2.47.1
Wine Mono Runtime	The Wine Project	5.0.0
Wine Mono Windows Support	The Wine Project	5.0.0

This will run the applications built-in installer, which should also have the option to uninstall.



Uninstall Notepad++ v7.7.1

Remove Notepad++ v7.7.1 from your computer.

Notepad++ v7.7.1 will be uninstalled from the following folder. Click Uninstall to start the uninstallation.

Uninstalling from: C:\Program Files\Notepad++\

The best things in life are free. Notepad++ is free. So Notepad++ is the best

13.6.2. Mizutamari

Mizutamari is an application similar to `winetricks`, although it was inspired by the Lutris gaming system for Linux. But while it is focused on games, there are also non-gaming applications available for install through Mizutamari.

Installing Mizutamari

To install Mizutamari's binary package, issue the following command:

```
# pkg install mizuma
```

Mizutamari is available in the FreeBSD Ports system. However, instead of looking in the *emulators* section of Ports or binary packages, look for it in the *games* section.

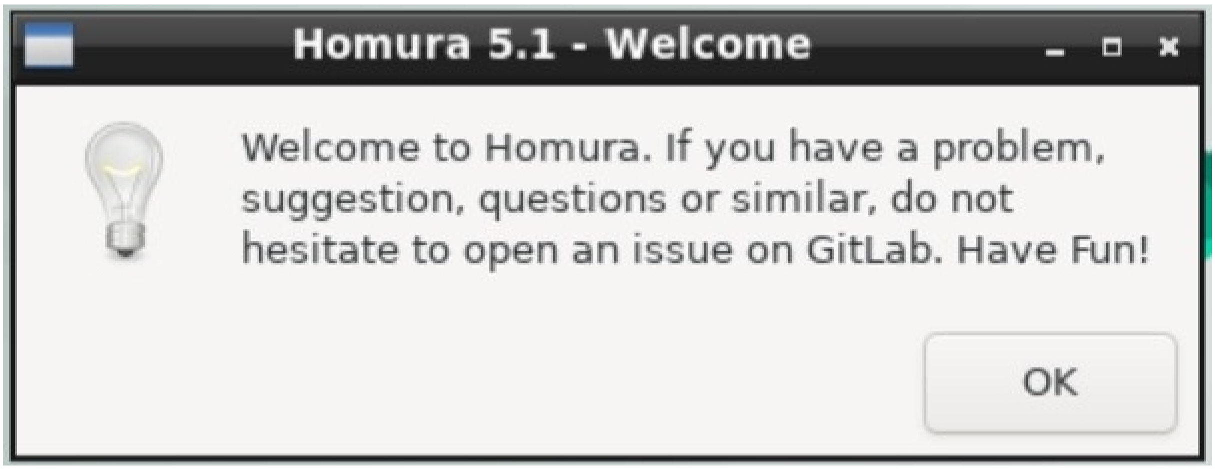
```
# cd /usr/ports/games/mizuma
# make install
```

Using Mizutamari

Mizutamari's usage is quite similar to that of `winetricks`. When using it for the first time, launch it from the command line (or a desktop environment runner applet) with:

```
% Mizuma
```

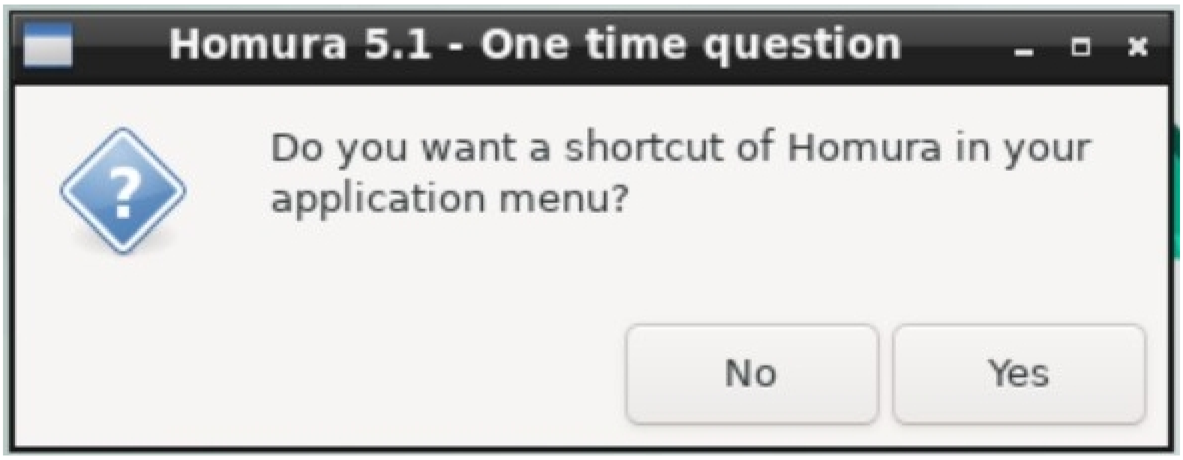
This should result in a friendly welcome message. Click *OK* to continue.



Homura 5.1 - Welcome

Welcome to Homura. If you have a problem, suggestion, questions or similar, do not hesitate to open an issue on GitLab. Have Fun!

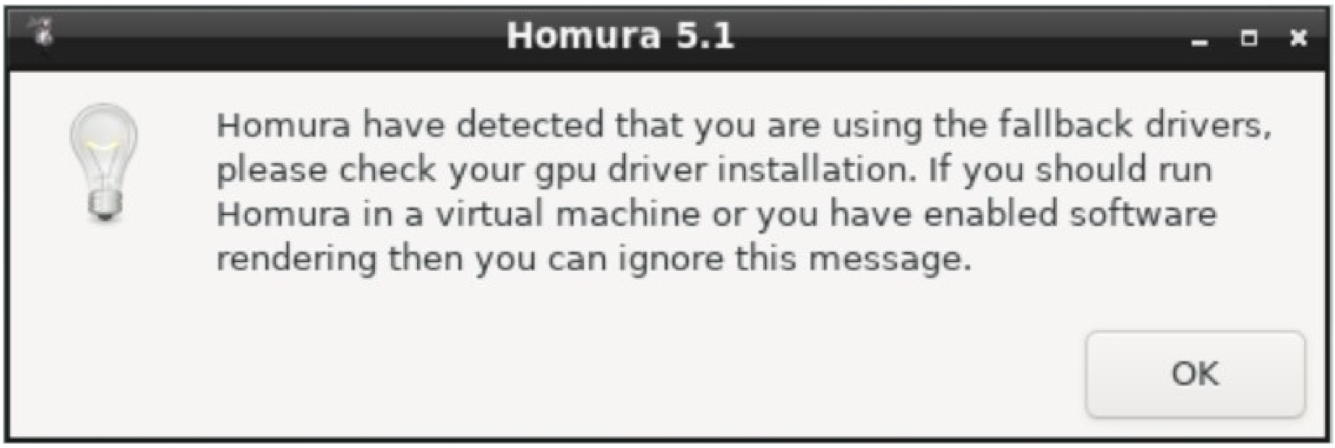
The program will also offer to place a link in the application menu of compatible environments:



Homura 5.1 - One time question

Do you want a shortcut of Homura in your application menu?

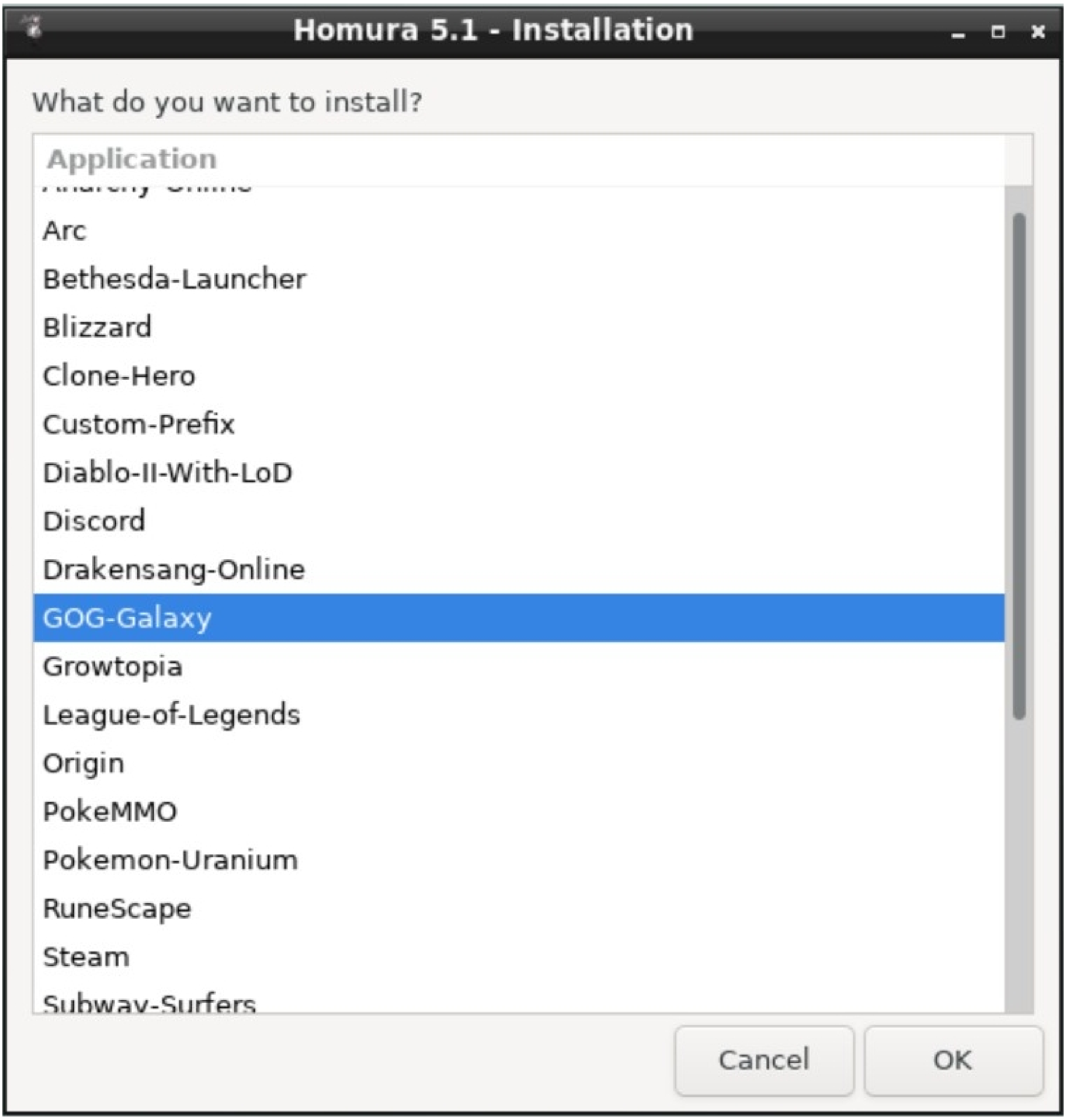
Depending on the setup of the FreeBSD machine, Mizutamari may display a message urging the install of native graphics drivers.



Homura 5.1

Homura have detected that you are using the fallback drivers, please check your gpu driver installation. If you should run Homura in a virtual machine or you have enabled software rendering then you can ignore this message.

The application’s window should then appear, which amounts to a "main menu" with all its options. Many of the items are the same as `winetricks`, although Mizutamari offers some additional, helpful options such as opening its data folder (*Open Mizutamari Folder*) or running a specified program (*Run a executable in prefix*).

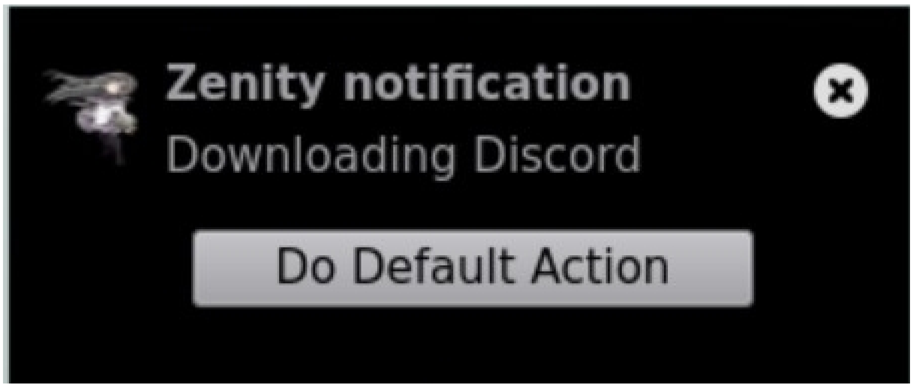


Homura 5.1 - Installation

What do you want to install?

Application
Arc
Bethesda-Launcher
Blizzard
Clone-Hero
Custom-Prefix
Diablo-II-With-LoD
Discord
Drakensang-Online
GOG-Galaxy
Growtopia
League-of-Legends
Origin
PokeMMO
Pokemon-Uranium
RuneScape
Steam
Subway-Surfers
Cancel
OK

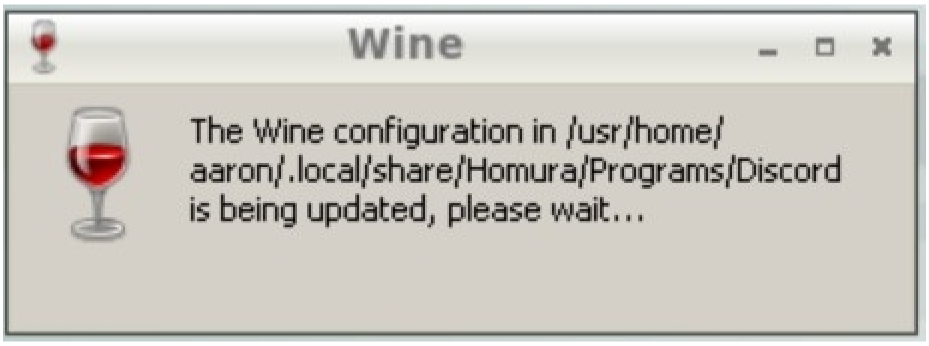
To select one of Mizutamari's supported applications to install, select *Installation*, and click *OK*. This will display a list of applications Homura can install automatically. Select one, and click *OK* to start the process.



Zenity notification

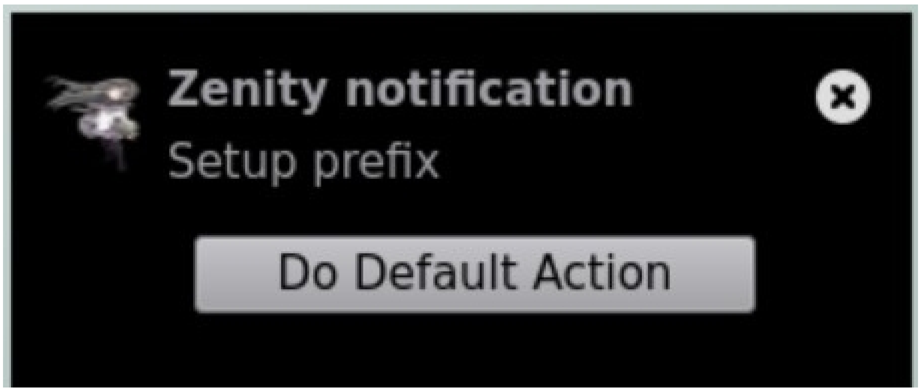
Downloading Discord

As a first step Mizutamari will download the selected program. A notification may appear in supported desktop environments.



The Wine configuration in /usr/home/aaron/.local/share/Homura/Programs/Discord is being updated, please wait...

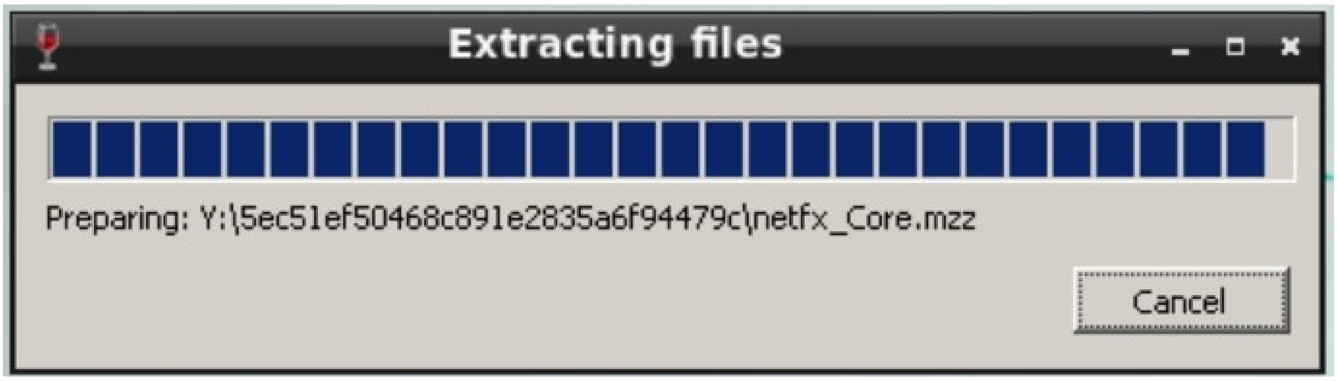
The program will also create a new prefix for the application. A standard WINE dialog with this message will display.



Zenity notification

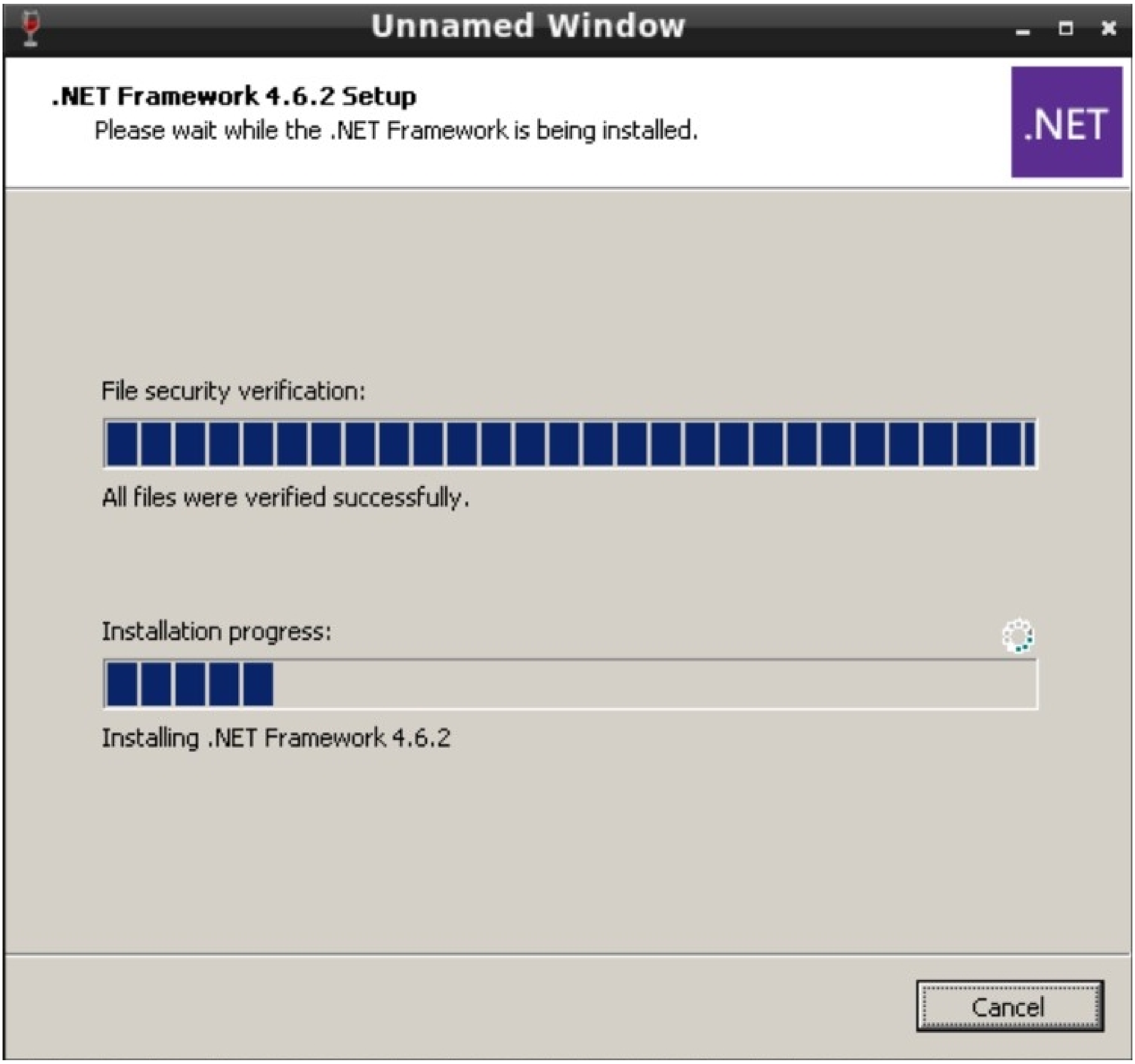
Setup prefix

Next, Mizutamari will install any prerequisites for the selected program. This may involve downloading and extracting a fair number of files, the details of which will show in dialogs.



Preparing: Y:\5ec51ef50468c891e2835a6f94479c\netfx_Core.mzz

Downloaded packages are automatically opened and run as required.



.NET Framework 4.6.2 Setup

Please wait while the .NET Framework is being installed.

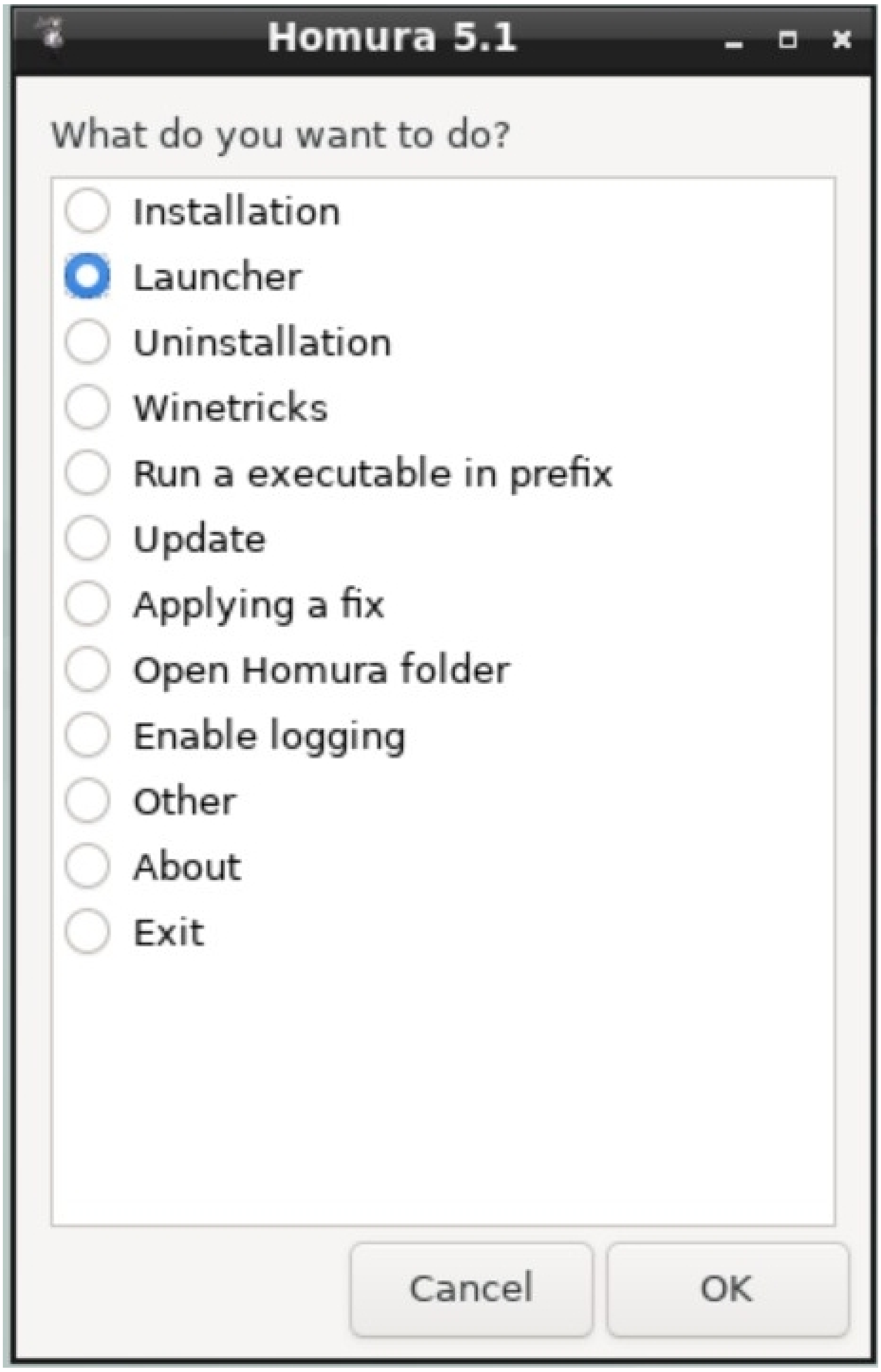
File security verification:

All files were verified successfully.

Installation progress:

Installing .NET Framework 4.6.2

The installation may end with a simple desktop notification or message in the terminal, depending on how Mizutamari was launched. But in either case Mizutamari should return to the main screen. To confirm the installation was successful, select *Launcher*, and click *OK*.

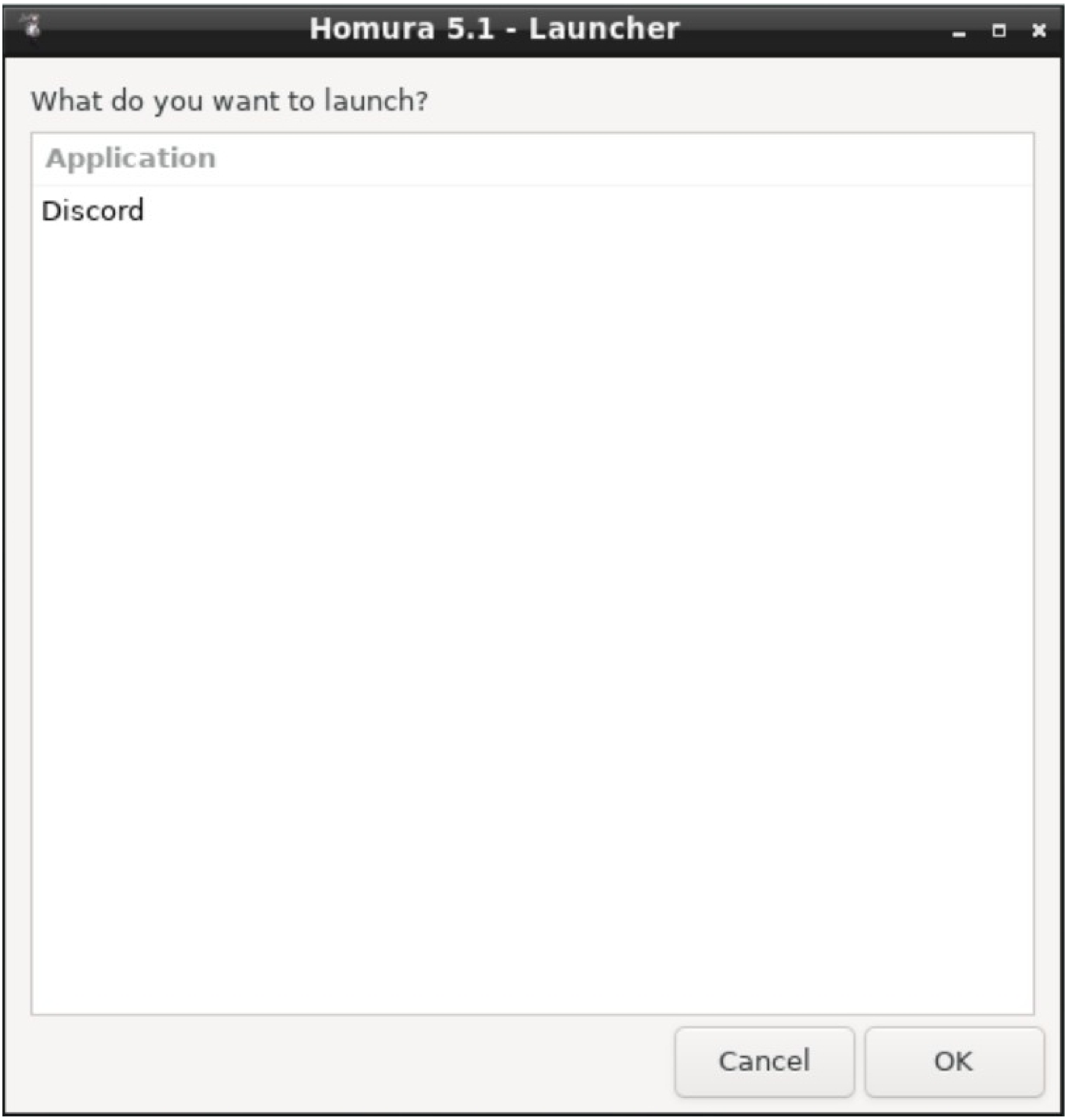


Homura 5.1

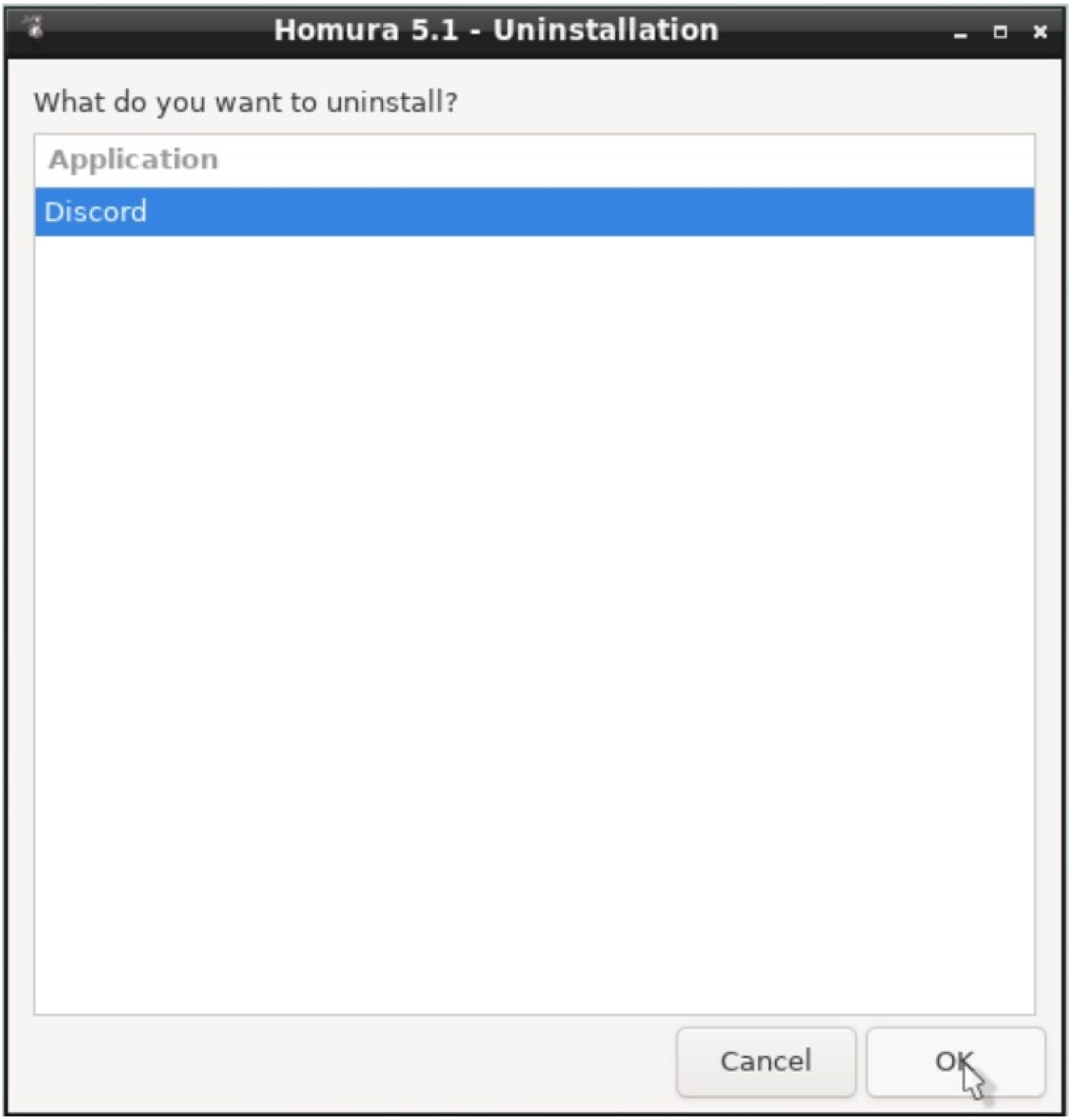
What do you want to do?

○ Installation
● Launcher
○ Uninstallation
○ Winetricks
○ Run a executable in prefix
○ Update
○ Applying a fix
○ Open Homura folder
○ Enable logging
○ Other
○ About
○ Exit

This will display a list of installed applications.


Homura 5.1 - Launcher
What do you want to launch?
Application
Discord
Cancel
OK

To run the new program, select it from the list, and click *OK*. To uninstall the application, select *Uninstallation* from the main screen, which will display a similar list. Select the program to be removed, and click *OK*.


Homura 5.1 - Uninstallation
What do you want to uninstall?
Application
Discord
Cancel
OK

13.6.3. Running Multiple Management GUIs

it is worth noting that the above solutions are not mutually exclusive. it is perfectly acceptable, even advantageous, to have both installed at the same time, as they support a different set of programs.

However, it is wise to ensure that they do not access any of the same WINE prefixes. Each of these solutions applies workarounds and makes changes to the registries based on known workarounds to existing WINE issues in order to make a given application run smoothly. Allowing both `winetricks` and Homura to access the same prefix could lead to some of these being overwritten, with the result being some or all applications do not work as expected.

13.7. WINE in Multi-User FreeBSD Installations

13.7.1. Issues with Using a Common WINE Prefix

Like most UNIX®-like operating systems, FreeBSD is designed for multiple users to be logged in and working at the same time. On the other hand, Windows® is multi-user in the sense that there can be multiple user accounts set up on one system. But the expectation is that only one will be using the physical machine (a desktop or laptop PC) at any given moment.

More recent consumer versions of Windows® have taken some steps to improve the OS in multi-user scenarios. But it is still largely structured around a single-user experience. Furthermore, the measures the WINE project has taken to create a compatible environment means, unlike FreeBSD applications (including WINE itself), it will resemble this single-user environment.

So it follows that each user will have to maintain their own set of configurations, which is potentially good. Yet it is advantageous to install applications, particularly large ones like office suites or games, only once. Two examples of reasons to do this are maintenance (software updates need only be applied once) and efficiency in storage (no duplicated files).

There are two strategies to minimize the impact of multiple WINE users in the system.

13.7.2. Installing Applications to a Common Drive

As shown in the section on WINE Configuration, WINE provides the ability to attach additional drives to a given prefix. In this way, applications can be installed to a common location, while each user will still have a prefix where individual settings may be kept (depending on the program). This is a good setup if there are relatively few applications to be shared between users, and they are programs that require few custom tweaks changes to the prefix in order to function.

The steps to make install applications in this way are as follows:

1. First, set up a shared location on the system where the files will be stored, such as /mnt/windows-drive_d/. Creating new directories is described in the mkdir(1) manual page.
2. Next, set permissions for this new directory to allow only desired users to access it. One approach to this is to create a new group such as "windows," add the desired users to that group (see the sub-section on groups in the Users and Basic Account Management section), and set to the permissions on the directory to `770` (the section on Permissions illustrates this process).
3. Finally, add the location as a drive to the user's prefix using the `winecfg` as described in the above section on WINE Configuration in this chapter.

Once complete, applications can be installed to this location, and subsequently run using the assigned drive letter (or the standard UNIX®-style directory path). However, as noted above, only one user should be running these applications (which may be accessing files within their installation directory) at the same time. Some applications may also exhibit unexpected behavior when run by a user who is not the owner, despite being a member of the group that should have full "read/write/execute" permissions for the entire directory.

13.7.3. Using a Common Installation of WINE

If, on the other hand, there are many applications to be shared, or they require specific tuning in order to work correctly, a different approach may be required. In this method, a completely separate user is created specifically for the purposes of storing the WINE prefix and all its installed applications. Individual users are then granted permission to run programs as this user using the sudo(8) command. The result is that these users can launch a WINE application as they normally would, only it will act as though launched by the newly-created user, and therefore use the centrally-maintained prefix containing both settings and programs. To accomplish this, take the following steps:

Create a new user with the following command (as `root`), which will step through the required details:

```
# adduser
```

Enter the username (e.g., *windows*) and Full name ("Microsoft Windows"). Then accept the defaults for the remainder of the questions. Next, install the `sudo` utility using binary packages with the following:

```
# pkg install sudo
```

Once installed, edit /etc/sudoers as follows:

```
# User alias specification

# define which users can run the wine/windows programs
User_Alias WINDOWS_USERS = user1,user2

# define which users can administrate (become root)
User_Alias ADMIN = user1

# Cmnd alias specification

# define which commands the WINDOWS_USERS may run
Cmnd_Alias WINDOWS = /usr/bin/wine,/usr/bin/winecfg

# Defaults
Defaults:WINDOWS_USERS env_reset
Defaults:WINDOWS_USERS env_keep += DISPLAY
Defaults:WINDOWS_USERS env_keep += XAUTHORITY
Defaults    !lecture,tty_tickets,!fqdn

# User privilege specification
root    ALL=(ALL) ALL

# Members of the admin user_alias, defined above, may gain root privileges
```

```
ADMIN ALL=(ALL) ALL

# The WINDOWS_USERS may run WINDOWS programs as user windows without a password
WINDOWS_USERS ALL = (windows) NOPASSWD: WINDOWS
```

The result of these changes is the users named in the *User_Alias* section are permitted to run the programs listed in the *Cmnd Alias* section using the resources listed in the *Defaults* section (the current display) as if they were the user listed in the final line of the file. In other words, users designates as *WINDOWS_USERS* can run the WINE and `winecfg` applications as user *windows*. As a bonus, the configuration here means they will not be required to enter the password for the *windows* user.

Next provide access to the display back to the *windows* user, as whom the WINE programs will be running:

```
% xhost +local:windows
```

This should be added to the list of commands run either at login or when the default graphical environment starts. Once all the above are complete, a user configured as one of the `WINDOW_USERS` in sudoers can run programs using the shared prefix with the following command:

```
% sudo -u windows wine program.exe
```

it is worth noting that multiple users accessing this shared environment at the same time is still risky. However, consider also that the shared environment can itself contain multiple prefixes. In this way an administrator can create a tested and verified set of programs, each with its own prefix. At the same time, one user can play a game while another works with office programs without the need for redundant software installations.

13.8. WINE on FreeBSD FAQ

The following section describes some frequently asked questions, tips/tricks, or common issues in running WINE on FreeBSD, along with their respective answers.

13.8.1. Basic Installation and Usage

How to Install 32-bit and 64-bit WINE on the Same System?

As described earlier in this section, the wine and i386-wine packages conflict with one another, and therefore cannot be installed on the same system in the normal way. However, multiple installs can be achieved using mechanisms like chroots/jails, or by building WINE from source (note this does *not* mean building the port).

Can DOS Programs Be Run on WINE?

They can, as "Console User Interface" applications as mentioned earlier in this section. However, there is an arguably better method for running DOS software: emulators/dosbox. On the other hand, there is little reason not to at least try it. Simply create a new prefix, install the software, and if it does not work delete the prefix.

Should the emulators/wine-devel Package/Port be Installed to Use the Development Version of WINE Instead of Stable?

Yes, installing this version will install the "development" version of WINE. As with the 32- and 64-bit versions, they cannot be installed together with the stable versions unless additional measures are taken.

Note that WINE also has a "Staging" version, which contains the most recent updates. This was at one time available as a FreeBSD port; however, it has since been removed. It can be compiled directly from source however.

13.8.2. Install Optimization

How Should Windows® Hardware (e.g., Graphics) Drivers be Handled?

Operating system drivers transfer commands between applications and hardware. WINE emulates a Windows® environment, including the drivers, which in turn use FreeBSD's native drivers for this transfer. it is not advisable to install Windows® drivers, as the WINE system is designed to use the host systems drivers. If, for example, a graphics card that benefits from dedicated drivers, install them using the standard FreeBSD methods, not Windows® installers.

Is There a way to Make Windows® Fonts Look Better?

A user on the FreeBSD forums suggests this configuration to fix out-of-the-box look of WINE fonts, which can be slightly pixelated.

According to a post in the FreeBSD Forums, adding the following to .config/fontconfig/fonts.conf will add anti-aliasing and make text more readable.

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd>"

<fontconfig>

  <!-- antialias all fonts -->
  <match target="font">
    <edit name="antialias" mode="assign"><bool>true</bool></edit>>
    <edit name="hinting" mode="assign"><bool>true</bool></edit>>
    <edit name="hintstyle" mode="assign"><const>hintslight</const></edit>>
    <edit name="rgba" mode="assign"><const>rgb</const></edit>>
  </match>
</fontconfig>
```

Does Having Windows® Installed Elsewhere on a System Help WINE Operate?

It may, depending on the application being run. As mentioned in the section describing `winecfg`, some built-in WINE DLLs and other libraries can be overridden by providing a path to an alternate version. Provided the Windows® partition or drive is mounted to the FreeBSD system and accessible to the user, configuring some of these overrides will use native Windows® libraries and may decrease the chance of unexpected behavior.

13.8.3. Application-Specific

Where is the Best Place to see if Application X Works on WINE?

The first step in determining compatibility should be the WINE AppDB. This is a compilation of reports of programs working (or not) on all supported platforms, although (as previously mentioned), solutions for one platform are often applicable to others.

Is There Anything That Will Help Games Run Better?

Perhaps. Many Windows® games rely on DirectX, a proprietary Microsoft graphics layer. However there are projects in the open source community attempting to implement support for this technology.

The *dxvk* project, which is an attempt to implement DirectX using the FreeBSD-compatible Vulkan graphics sub-system, is one such. Although its primary target is WINE on Linux, some FreeBSD users report compiling and using dxvk.

In addition, work is under way on a wine-proton port. This will bring the work of Valve, developer of the Steam gaming platform, to FreeBSD. Proton is a distribution of WINE designed to allow many Windows® games to run on other operating systems with minimal setup.

Is There Anywhere FreeBSD WINE Users Gather to Exchange Tips and Tricks?

There are plenty of places FreeBSD users discuss issues related to WINE that can be searched for solutions:

- The FreeBSD forums, particularly the *Installation and Maintenance of Ports or Packages* or *Emulation and virtualization* forums.
- FreeBSD IRC channels including #freebsd (for general support), #freebsd-games, and others.
- The BSD World Discord server's channels including *bsd-desktop*, *bsd-gaming*, *bsd-wine*, and others.

13.8.4. Other OS Resources

There are a number of resources focused on other operating systems that may be useful for FreeBSD users:

- The WINE Wiki has a wealth of information on using WINE, much of which is applicable across many of WINE's supported operating systems.
- Similarly, the documentation available from other OS projects can also be of good value. The

WINE page on the Arch Linux Wiki is a particularly good example, although some of the "Third-party applications" (i.e., "companion applications") are obviously not available on FreeBSD.

- Finally, Codeweavers (a developer of a commercial version of WINE) is an active upstream contributor. Oftentimes answers to questions in their support forum can be of aid in troubleshooting problems with the open source version of WINE.

Part III: System Administration

The remaining chapters cover all aspects of FreeBSD system administration. Each chapter starts by describing what will be learned as a result of reading the chapter, and also details what the reader is expected to know before tackling the material.

These chapters are designed to be read as the information is needed. They do not need to be read in any particular order, nor must all of them be read before beginning to use FreeBSD.

Chapter 14. Configuration, Services, Logging and Power Management

14.1. Synopsis

One of the important aspects of FreeBSD is proper system configuration. This chapter explains much of the FreeBSD configuration process, including some of the parameters which can be set to tune a FreeBSD system.

Before reading this chapter:

- Understand UNIX® and FreeBSD basics (FreeBSD Basics).

Read this chapter to learn:

- How to use the various configuration files in /etc.
- The basics of rc.conf configuration and /usr/local/etc/rc.d startup scripts.
- How to tune FreeBSD using sysctl(8) variables.
- How to configure the power management in FreeBSD.

14.2. Configuration Files

FreeBSD maintains a clear separation between the base system and third party applications and therefore this affects where the configuration files of these applications are located.

FreeBSD base system configuration is located at the /etc directory, and the /usr/local/etc directory contains all the configuration files of the applications installed on the system through the ports collection and packages.

The kernel state configuration is located in /etc/sysctl.conf. In the section The sysctl utility, the operation of sysctl(8) will be explained in more detail.

For more information about the FreeBSD file system structure refer to hier(7).

As a general rule, configuration files do not use a standard on what syntax they must follow. Although it is true that the `#` character is normally used to comment a line and that each line has a configuration variable.

> Some applications like pkg(8) are starting to use the Universal Configuration Language (UCL).

14.2.1. The /etc directory

The /etc directory contains all of the FreeBSD base system configuration files that are responsible for configuring FreeBSD.

Extreme caution must be taken when modifying files in the /etc directory; misconfiguration could make FreeBSD unbootable or malfunction.

/etc	System configuration files and scripts.
/etc/defaults	Default system configuration files, see rc(8) for more information.
/etc/fstab	fstab(5) contains descriptive information about the various file systems.
/etc/mail	Extra sendmail(8) configuration and other MTA configuration files.
/etc/mtree	mtree configuration files, see mtree(8) for more information.
/etc/pam.d	Configuration files for the Pluggable Authentication Modules (PAM) library.
/etc/periodic	Scripts that are run daily, weekly, and monthly, via cron(8), see periodic(8) for more information.
/etc/rc.d	System and daemon startup/control scripts, see rc(8) for more information.
/etc/rc.conf	Contains descriptive information about the local host name, configuration details for any potential network interfaces and which services should be started up at system initial boot time. More information in Managing System-Specific Configuration
/etc/security	OpenBSM audit configuration files, see audit(8) for more information.
/etc/ppp	ppp configuration files, see ppp(8) for more information.
/etc/ssh	OpenSSH configuration files, see ssh(1) for more information.
/etc/ssl	OpenSSL configuration files.
/etc/sysctl.conf	Contains settings for the kernel. More information in The sysctl utility

14.2.2. The sysctl utility

The sysctl(8) utility is used to make changes to a running FreeBSD system.

The sysctl(8) utility retrieves kernel state and allows processes with appropriate privilege to set kernel state. The state to be retrieved or set is described using a "Management Information Base" ("MIB") style name, described as a dotted set of components.

Table 24. Management Information Base

sysctl	"Magic" numbers
kern	Kernel functions and features
vm	Virtual memory
vfs	Filesystem
net	Network
debug	Debugging parameters
hw	Hardware
machdep	Machine dependent
user	Userland
p1003_1b	POSIX 1003.1B

At its core, sysctl(8) serves two functions: to read and to modify system settings.

To view all readable variables:

```
% sysctl -a
```

The output should be similar to the following:

```
kern.ostype: FreeBSD
...
vm.swap_enabled: 1
vm.overcommit: 0
vm.domain.0.pidctrl.kdd: 8
vm.domain.0.pidctrl.kid: 4
vm.domain.0.pidctrl.kpd: 3
...
vfs.zfs.sync_pass_rewrite: 2
vfs.zfs.sync_pass_dont_compress: 8
vfs.zfs.sync_pass_deferred_free: 2
```

To read a particular variable, specify its name:

```
% sysctl kern.maxproc
```

The output should be similar to the following:

```
kern.maxproc: 1044
```

The Management Information Base (MIB) is hierarchical and hence, specifying a prefix prints all the nodes hanging from it:

```
% sysctl net
```

The output should be similar to the following:

```
net.local.stream.recvspace: 8192
net.local.stream.sendspace: 8192
net.local.dgram.recvspace: 16384
net.local.dgram.maxdgram: 2048
net.local.seqpacket.recvspace: 8192
net.local.seqpacket.maxseqpacket: 8192
net.local.sockcount: 60
net.local.taskcount: 25
net.local.recycled: 0
net.local.deferred: 0
net.local.inflight: 0
net.inet.ip.portrange.randomtime: 1
net.inet.ip.portrange.randomcps: 9999
[...]
```

To set a particular variable, use the *variable*=*value* syntax:

```
# sysctl kern.maxfiles=5000
```

The output should be similar to the following:

```
kern.maxfiles: 2088 -> 5000
```

To keep the configuration after a reboot it is necessary to add these variables to the /etc/sysctl.conf file as explained below.

14.2.3. The /etc/sysctl.conf file

The configuration file for sysctl(8), /etc/sysctl.conf, looks much like /etc/rc.conf.

Values are set using a `variable=value` syntax.

The specified values are set after the system goes into multi-user mode. Not all variables are settable in this mode.

For example, to turn off logging of fatal signal exits and prevent users from seeing processes started by other users, the following tunables can be set in /etc/sysctl.conf:

```
# Do not log fatal signal exits (e.g., sig 11)
```

```
kern.logsigexit=0

# Prevent users from seeing information about processes that
# are being run under another UID.
security.bsd.see_other_uids=0
```

To obtain more information about what function a particular sysctl has, the following command can be executed:

```
% sysctl -d kern.dfldsiz
```

The output should be similar to the following:

```
kern.dfldsiz: Initial data size limit
```

14.2.4. Managing System-Specific Configuration

The principal location for system configuration information is /etc/rc.conf.

This file contains a wide range of configuration information and it is read at system startup to configure the system. It provides the configuration information for the rc* files.

The entries in /etc/rc.conf override the default settings in /etc/defaults/rc.conf.

> The file /etc/defaults/rc.conf containing the default settings should not be edited. Instead, all system-specific changes should be made to /etc/rc.conf.

A number of strategies may be applied in clustered applications to separate site-wide configuration from system-specific configuration in order to reduce administration overhead.

The recommended approach is to place system-specific configuration into /etc/rc.conf.local.

For example, these entries in /etc/rc.conf apply to all systems:

```
sshd_enable="YES"
keyrate="fast"
defaultrouter="10.1.1.254"
```

Whereas these entries in /etc/rc.conf.local apply to this system only:

```
hostname="node1.example.org"
ifconfig_fxp0="inet 10.1.1.1/8"
```

Distribute /etc/rc.conf to every system using an application such as rsync or puppet, while /etc/rc.conf.local remains unique.

Upgrading the system will not overwrite /etc/rc.conf, so system configuration information will not be lost.

Both /etc/rc.conf and /etc/rc.conf.local are parsed by sh(1). This allows system operators to create complex configuration scenarios. Refer to rc.conf(5) for further information on this topic.

14.3. Managing Services in FreeBSD

FreeBSD uses the rc(8) system of startup scripts during system initialization and for managing services.

The scripts listed in /etc/rc.d provide basic services which can be controlled with the `start`, `stop`, and `restart` options to service(8).

A basic script may look similar to the following:

```
#!/bin/sh
#
# PROVIDE: utility
# REQUIRE: DAEMON
# KEYWORD: shutdown

. /etc/rc.subr

name=utility
rcvar=utility_enable

command="/usr/local/sbin/utility"

load_rc_config $name

#
# DO NOT CHANGE THESE DEFAULT VALUES HERE
# SET THEM IN THE /etc/rc.conf FILE
#
utility_enable=${utility_enable-"NO"}
pidfile=${utility_pidfile-"/var/run/utility.pid"}

run_rc_command "$1"
```

Refer to this article for instructions on how to create custom rc(8) scripts.

14.3.1. Starting Services

Many users install third party software on FreeBSD from the Ports Collection and require the installed services to be started upon system initialization.

Services, such as security/openssh-portable or www/nginx are just two of the many software packages which may be started during system initialization. This section explains the procedures available for starting services.

Since the rc(8) system is primarily intended to start and stop services at system startup and shutdown time, the `start`, `stop` and `restart` options will only perform their action if the appropriate /etc/rc.conf variable is set.

So the first step to start a service, like for example www/nginx is to add it to /etc/rc.conf by executing the following command:

```
# sysrc nginx_enable="YES"
```

Then nginx can be started executing the following command:

```
# service nginx start
```

To `start`, `stop` or `restart` a service regardless of the settings in /etc/rc.conf, these commands should be prefixed with "one". For instance, to start www/nginx regardless of the current /etc/rc.conf setting, execute the following command:

```
# service nginx onestart
```

It is also possible to put a service automatically into a jail, see the corresponding Service Jails explanation.

14.3.2. Status of a Service

To determine if a service is running, use the `status` subcommand.

For example, to verify that www/nginx is running:

```
# service nginx status
```

The output should be similar to the following:

```
nginx is running as pid 27871.
```

14.3.3. Reload a Service

In some cases, it is also possible to `reload` a service. This attempts to send a signal to an individual service, forcing the service to reload its configuration files.

In most cases, this means sending the service a `SIGHUP` signal.

Not all services support this feature.

The rc(8) system is used for network services and it also contributes to most of the system initialization. For instance, when the /etc/rc.d/bgfsck script is executed, it prints out the following message:

```
Starting background file system checks in 60 seconds.
```

This script is used for background file system checks, which occur only during system initialization.

Many system services depend on other services to function properly. For example, yp(8) and other RPC-based services may fail to start until after the rpcbind(8) service has started.

Additional information can be found in rc(8) and rc.subr(8).

14.3.4. Using Services to Start Services

Other services can be started using inetd(8). Working with inetd(8) and its configuration is described in depth in "The inetd Super-Server".

In some cases, it may make more sense to use cron(8) to start system services. This approach has a number of advantages as cron(8) runs these processes as the owner of the crontab(5). This allows regular users to start and maintain their own applications.

The `@reboot` feature of cron(8), may be used in place of the time specification. This causes the job to run when cron(8) is started, normally during system initialization.

14.4. Cron and Periodic

Scheduling tasks to run at a certain day or time is a very common task on FreeBSD. The tool in charge of performing this task is cron(8).

In addition to tasks that can be scheduled by the user via cron(8), FreeBSD performs routine background tasks managed by periodic(8).

14.4.1. Cron

The cron(8) utility runs in the background and regularly checks /etc/crontab for tasks to execute and searches /var/cron/tabs for custom crontab files.

These files are used to schedule tasks which cron runs at the specified times.

Each entry in a crontab defines a task to run and is known as a *cron job*.

Two different types of configuration files are used: the system crontab, which should not be modified, and user crontabs, which can be created and edited as needed. The format used by these files is documented in crontab(5). The format of the system crontab, /etc/crontab includes a `who` column which does not exist in user crontabs. In the system crontab, cron runs the command as the user specified in this column. In a user crontab, all commands run as the user who created the crontab.

User crontabs allow individual users to schedule their own tasks. The `root` user can also have a user crontab which can be used to schedule tasks that do not exist in the system crontab.

Here is a sample entry from the system crontab, /etc/crontab:

```
# /etc/crontab - root's crontab for FreeBSD
#
①
#
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin ②
#
#minute hour    mday    month   wday    who     command ③
#
# Save some entropy so that /dev/random can re-seed on boot.
*/11    *       *       *       *       operator /usr/libexec/save-entropy ④
#
# Rotate log files every hour, if necessary.
0       *       *       *       *       root    newsyslog
#
# Perform daily/weekly/monthly maintenance.
1       3       *       *       *       root    periodic daily
15      4       *       *       6       root    periodic weekly
30      5       1       *       *       root    periodic monthly
#
# Adjust the time zone if the CMOS clock keeps local time, as opposed to
# UTC time.  See adjkerntz(8) for details.
1,31    0-5     *       *       *       root    adjkerntz -a
```

① Lines that begin with the `#` character are comments. A comment can be placed in the file as a reminder of what and why a desired action is performed. Comments cannot be on the same line as a command or else they will be interpreted as part of the command; they must be on a new line. Blank lines are ignored.

② The equals (`=`) character is used to define any environment settings. In this example, it is used to define the `SHELL` and `PATH`. If the `SHELL` is omitted, cron will use the default POSIX® shell. If the `PATH` is omitted, the full path must be given to the command or script to run.

③ This line defines the seven fields used in a system crontab: `minute`, `hour`, `mday`, `month`, `wday`, `who`, and `command`. The `minute` field is the time in minutes when the specified command will be run, the `hour` is the hour when the specified command will be run, the `mday` is the day of the month, `month` is the month, and `wday` is the day of the week. These fields must be numeric values, representing the twenty-four hour clock, or a `*`, representing all values for that field. The `who` field only exists in the system crontab and specifies which user the command should be run as. The last field is the command to be executed.

④ This entry defines the values for this cron job. The `*/11`, followed by several more `*` characters, specifies that `/usr/libexec/save-entropy` is invoked by `operator` every eleven minutes of every hour, of every day and day of the week, of every month. Commands can include any number of switches. However, commands which extend to multiple lines need to be broken with the backslash "\" continuation character.

14.4.2. Creating a User Crontab

To create a user crontab, invoke `crontab` in editor mode:

```
% crontab -e
```

This will open the user's crontab using the default text editor. The first time a user runs this command, it will open an empty file. Once a user creates a crontab, this command will open that file for editing.

It is useful to add these lines to the top of the crontab file in order to set the environment variables and to remember the meanings of the fields in the crontab:

```
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin
# Order of crontab fields
# minute hour mday month wday command
```

Then add a line for each command or script to run, specifying the time to run the command. This example runs the specified custom shell script every day at two in the afternoon. Since the path to the script is not specified in `PATH`, the full path to the script is given:

```
0 14 * * * /home/user/bin/mycustomscript.sh
```

Before using a custom script, make sure it is executable and test it with the limited set of environment variables set by cron. To replicate the environment that would be used to run the above cron entry, use:

```
env -i SHELL=/bin/sh PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
HOME=/home/user LOGNAME=user /home/user/bin/mycustomscript.sh
```

The environment set by cron is discussed in crontab(5). Checking that scripts operate correctly in a cron environment is especially important if they include any commands that delete files using wildcards.

When finished editing the crontab, save the file. It will automatically be installed, and cron will read the crontab and run its cron jobs at their specified times. To list the cron jobs in a crontab, use this command:

```
% crontab -l
```

The output should be similar to the following:

```
0 14 * * * /home/user/bin/mycustomscript.sh
```

To remove all of the cron jobs in a user crontab:

```
% crontab -r
```

The output should be similar to the following:

```
remove crontab for user? y
```

14.4.3. Periodic

FreeBSD provides a set of system management scripts to check status of various subsystems, perform security-related checks, rotate log files, etc. These scripts are run on a periodic basis: daily. weekly, or monthly. The management of these tasks is performed by periodic(8) and its configuration resides in periodic.conf(5). The periodic tasks are initiated by entries in the system crontab, shown above.

Scripts executed by periodic(8) are located in /etc/periodic/ for base utilities and in /usr/local/etc/periodic/ for third-party software.

They are organized in 4 subdirectories, daily, weekly, monthly and security.

14.4.4. Enable or Disable Periodic Tasks

FreeBSD has some scripts enabled by default to run periodically.

To enable or disable a task, the first step is to edit /etc/periodic.conf executing the following command:

```
# ee /etc/periodic.conf
```

And then to enable, for example, `daily_status_zfs_enable` put the following content in the file:

```
daily_status_zfs_enable="YES"
```

To disable a task that is active by default, all that needs to be done is to change `YES` to `NO`.

14.4.5. Configuring the Output of Periodic Tasks

In /etc/periodic.conf the variables `daily_output`, `weekly_output` and `monthly_output` specifies where to send the results of the script execution.

By default the output of the periodic scripts are emailed to root, and therefore it is best to read root's mail or alias root to a mailbox that is monitored.

To send the results to another email or to other emails, add the email addresses separated by spaces to /etc/periodic.conf:

```
daily_output="email1@example.com email2@example.com"
weekly_output="email1@example.com email2@example.com"
monthly_output="email1@example.com email2@example.com"
```

To log periodic output instead of receiving it as email, add the following lines to /etc/periodic.conf. newsyslog(8) will rotate these files at the appropriate times:

```
daily_output=/var/log/daily.log
weekly_output=/var/log/weekly.log
monthly_output=/var/log/monthly.log
```

14.5. Configuring System Logging

Generating and reading system logs is an important aspect of system administration. The information in system logs can be used to detect hardware and software issues as well as application and system configuration errors. This information also plays an important role in security auditing and incident response. Most system daemons and applications will generate log entries.

FreeBSD provides a system logger, syslogd(8), to manage logging. By default, syslogd is enabled and started when the system boots.

This section describes how to configure the FreeBSD system logger for both local and remote logging and how to perform log rotation and log management.

14.5.1. Configuring Local Logging

The configuration file, /etc/syslog.conf, controls what syslogd does with log entries as they are received. There are several parameters to control the handling of incoming events. The *facility* describes which subsystem generated the message, such as the kernel or a daemon, and the *level* describes the severity of the event that occurred. This makes it possible to configure if and where a log message is logged, depending on the facility and level. It is also possible to take action depending on the application that sent the message, and in the case of remote logging, the hostname of the machine generating the logging event.

This configuration file contains one line per action, where the syntax for each line is a selector field followed by an action field. The syntax of the selector field is *facility.level* which will match log messages from *facility* at level *level* or higher. It is also possible to add an optional comparison flag before the level to specify more precisely what is logged. Multiple selector fields can be used for the same action, and are separated with a semicolon (`;`). Using `*` will match everything. The action field denotes where to send the log message, such as to a file or remote log host.

As an example, here is the default /etc/syslog.conf from FreeBSD:

```
#       Spaces ARE valid field separators in this file. However,
#       other *nix-like systems still insist on using tabs as field
#       separators. If you are sharing this file between systems, you
#       may want to use only tabs as field separators here.
#       Consult the syslog.conf(5) manpage.
*.err;kern.warning;auth.notice;mail.crit                /dev/console ①
*.notice;authpriv.none;kern.debug;lpr.info;mail.crit;news.err   /var/log/messages
security.*                                      /var/log/security
auth.info;authpriv.info                         /var/log/auth.log
mail.info                                       /var/log/maillog ②
cron.*                                          /var/log/cron
!-devd
*.=debug                                        /var/log/debug.log ③
*.emerg                                         *
daemon.info                                     /var/log/daemon.log
# uncomment this to log all writes to /dev/console to /var/log/console.log
# touch /var/log/console.log and chmod it to mode 600 before it will work
#console.info                                   /var/log/console.log
# uncomment this to enable logging of all log messages to /var/log/all.log
# touch /var/log/all.log and chmod it to mode 600 before it will work
#*.*                                            /var/log/all.log
# uncomment this to enable logging to a remote loghost named loghost
#*.*                                            @loghost
# uncomment these if you're running inn
# news.crit                                     /var/log/news/news.crit
# news.err                                      /var/log/news/news.err
# news.notice                                   /var/log/news/news.notice
# Uncomment this if you wish to see messages produced by devd
# !devd
# *.>=notice                                    /var/log/devd.log ④
!*
include                                         /etc/syslog.d
include                                         /usr/local/etc/syslog.d
```

① Matches all messages with a level of `err` or higher, as well as `kern.warning`, `auth.notice` and `mail.crit`, and sends these log messages to the console (/dev/console).

② Matches all messages from the `mail` facility at level `info` or above and logs the messages to /var/log/maillog.

③ Uses a comparison flag (`=`) to only match messages at level `debug` and logs them to /var/log/debug.log.

④ Is an example usage of a program specification. This makes the rules following it only valid for the specified program. In this case, only the messages generated by devd(8) are logged to /var/log/devd.log.

For more information about /etc/syslog.conf, its syntax, and more advanced usage examples, see syslog.conf(5).

14.5.2. Logging Facilities

A facility describes the part of the system generating the message. Facilities are a way of separating the different messages so that it is easier for the user to consult the logs.

Table 25. syslog facilities

Name	Description
auth	The authorization system: login(1), su(1), getty(8), etc.
authpriv	The same as auth, but logged to a file readable only by root.
console	Messages written to /dev/console by the kernel console output driver.
cron	Messages written by the cron(8) daemon.
daemon	System daemons, such as routed(8), that are not provided for explicitly by other facilities.
ftp	The file transfer protocol daemons: ftpd(8), tftpd(8).
kern	Messages generated by the kernel. These cannot be generated by any user processes.
lpr	The line printer spooling system: lpr(1), lpc(8), lpd(8), etc.
mail	The mail system.
mark	This facility adds a record every 20 minutes.
news	The network news system.
ntp	The network time protocol system.
security	Security subsystems, such as ipfw(4).
syslog	Messages generated internally by syslogd(8).
user	Messages generated by random user processes. **This is the default facility identifier if none is specified**.
uucp	The Unix-to-Unix Copy system. An ancient protocol. Really weird to see messages from this facility.
local0 through local7	Reserved for local use.

14.5.3. Logging Levels

The level describes the severity of the message, and is a keyword from the following ordered list (higher to lower):

Table 26. syslog levels

Name	Description
emerg	A panic condition. This is normally broadcast to all users.
alert	A condition that should be corrected immediately, such as a corrupted system database.
crit	Critical conditions, e.g., hard device errors.
err	Errors.
warning	Warning messages.
notice	Conditions that are not error conditions, but should possibly be handled specially.
info	Informational messages.
debug	Messages that contain information normally of use only when debugging a program.
none	This special level disables a particular facility.

14.5.4. Read Log Messages

By default FreeBSD log files use the format rfc3164, also known as The BSD syslog Protocol. Learn more about other formats and how to use them at syslog(8).

Typically the logs have the following syntax:

```
date time hostname program[pid]: the message
```

The output of the /var/log/cron file will be used as an example:

```
[...]
Jul 16 12:40:00 FreeBSD /usr/sbin/cron[81519]: (root) CMD (/usr/libexec/atrun)
Jul 16 12:44:00 FreeBSD /usr/sbin/cron[83072]: (operator) CMD (/usr/libexec/save-
entropy)
[...]
```

Verbose logging, so the facility and the level on each message will be added, can be enabled in syslog(8) by running the following command:

```
# sysrc syslogd_flags="-vv"
```

Once the function is activated, the facility and the level will be displayed in the log as shown in the following example:

```
[...]
Jul 16 17:40:00 <cron.info> FreeBSD /usr/sbin/cron[1016]: (root) CMD
(/usr/libexec/atrun)
Jul 16 17:44:00 <cron.info> FreeBSD /usr/sbin/cron[1030]: (operator) CMD
(/usr/libexec/save-entropy)
[...]
```

14.5.5. Log Management and Rotation

Log files can grow quickly, taking up disk space and making it more difficult to locate useful information.

In FreeBSD, newsyslog(8) is used to manage log files and attempt to mitigate this.

This built-in program periodically rotates and compresses log files, and optionally creates missing log files and signals programs when log files are moved.

> Since newsyslog is run from cron(8), it cannot rotate files more often than it is scheduled to run from cron(8). In the default configuration, it runs every hour.

Here is the default configuration in FreeBSD, more information in newsyslog.conf(5):

```
# configuration file for newsyslog
#
# Entries which do not specify the '/pid_file' field will cause the
# syslogd process to be signalled when that log file is rotated.  This
# action is only appropriate for log files which are written to by the
# syslogd process (ie, files listed in /etc/syslog.conf).  If there
# is no process which needs to be signalled when a given log file is
# rotated, then the entry for that file should include the 'N' flag.
#
# Note: some sites will want to select more restrictive protections than the
# defaults.  In particular, it may be desirable to switch many of the 644
# entries to 640 or 600.  For example, some sites will consider the
# contents of maillog, messages, and lpd-errs to be confidential.  In the
# future, these defaults may change to more conservative ones.
#
# logfilename          [owner:group]    mode count size when  flags [/pid_file]
[sig_num]
/var/log/all.log                        600  7     *    @T00  J
/var/log/auth.log                       600  7     1000 @0101T JC
/var/log/console.log                    600  5     1000 *     J
/var/log/cron                           600  3     1000 *     JC
/var/log/daily.log                      640  7     *    @T00  JN
/var/log/debug.log                      600  7     1000 *     JC
/var/log/init.log                       644  3     1000 *     J
/var/log/kerberos.log                   600  7     1000 *     J
/var/log/maillog                        640  7     *    @T00  JC
/var/log/messages                       644  5     1000 @0101T JC
```

```
/var/log/monthly.log                    640  12    *    $M1D0 JN
/var/log/devd.log                       644  3     1000 *     JC
/var/log/security                       600  10    1000 *     JC
/var/log/utx.log                        644  3     *    @01T05 B
/var/log/weekly.log                     640  5     *    $W6D0 JN
/var/log/daemon.log                     644  5     1000 @0101T JC

<include> /etc/newsyslog.conf.d/[!.]*.conf
<include> /usr/local/etc/newsyslog.conf.d/[!.]*.conf
```

1. `logfilename` - Name of the system log file to be archived.
2. `[owner:group]` - This optional field specifies the owner and group for the archive file.
3. `mode` - Specify the file mode of the log file and archives. Valid mode bits are 0666. (That is, read and write permissions for the rotated log may be specified for the owner, group, and others.)
4. `count` - Specify the maximum number of archive files which may exist.
5. `size` - When the size of the log file reaches size in kilobytes, the log file will be trimmed as described above. If this field contains an asterisk ('*'), the log file will not be trimmed based on size.
6. `when` - Consist of an interval, a specific time, or both. Supported options in newsyslog.conf(5).
7. `flags` - Indicates the flags that newsyslog accepts, supported options in newsyslog.conf(5).
8. `[/pid_file]` - This optional field specifies the file name containing a daemon's process ID or to find a group process ID.
9. `[sig_num]` - This optional field specifies the signal that will be sent to the daemon process.

> The last two fields are optional and specify the name of the Process ID (PID) file of a process and a signal number to send to that process when the file is rotated.

14.5.6. Configuring Remote Logging

Monitoring the log files of multiple hosts can become unwieldy as the number of systems increases. Configuring centralized logging can reduce some of the administrative burden of log file administration.

In FreeBSD, centralized log file aggregation, merging, and rotation can be configured using syslogd and newsyslog.

This section demonstrates an example configuration, where host `A`, named `logserv.example.com`, will collect logging information for the local network.

Host `B`, named `logclient.example.com`, will be configured to pass logging information to the logging server.

Log Server Configuration

A log server is a system that has been configured to accept logging information from other hosts.

Before configuring a log server, check the following:

- If there is a firewall between the logging server and any logging clients, ensure that the firewall ruleset allows UDP port 514 for both the clients and the server.
- The logging server and all client machines must have forward and reverse entries in the local DNS. If the network does not have a DNS server, create entries in each system's /etc/hosts. Proper name resolution is required so that log entries are not rejected by the logging server.

On the log server, edit /etc/syslog.conf to specify the name of the client to receive log entries from, the logging facility to be used, and the name of the log to store the host's log entries. This example adds the hostname of `B`, logs all facilities, and stores the log entries in /var/log/logclient.log.

Example 22. Sample Log Server Configuration

```
+logclient.example.com
*.*     /var/log/logclient.log
```

When adding multiple log clients, add a similar two-line entry for each client. More information about the available facilities may be found in syslog.conf(5).

Next, execute the following commands:

```
# sysrc syslogd_enable="YES"
# sysrc syslogd_flags="-a logclient.example.com -v -v"
```

The first entry starts syslogd at system boot. The second entry allows log entries from the specified client. The `-v -v` increases the verbosity of logged messages. This is useful for tweaking facilities as administrators are able to see what type of messages are being logged under each facility.

Multiple `-a` options may be specified to allow logging from multiple clients. IP addresses and whole netblocks may also be specified. Refer to syslogd(8) for a full list of possible options.

Finally, create the log file:

```
# touch /var/log/logclient.log
```

At this point, syslogd should be restarted and verified:

```
# service syslogd restart
# pgrep syslog
```

If a PID is returned, the server restarted successfully, and client configuration can begin. If the server did not restart, consult /var/log/messages for the error.

Log Client Configuration

A logging client sends log entries to a logging server on the network. The client also keeps a local copy of its own logs.

Once a logging server has been configured, execute the following commands on the logging client:

```
# sysrc syslogd_enable="YES"
# sysrc syslogd_flags="-s -v -v"
```

The first entry enables syslogd on boot up. The second entry prevents logs from being accepted by this client from other hosts (`-s`) and increases the verbosity of logged messages.

Next, define the logging server in the client's /etc/syslog.conf. In this example, all logged facilities are sent to a remote system, denoted by the `@` symbol, with the specified hostname:

```
*.*  @logserv.example.com
```

After saving the edit, restart syslogd for the changes to take effect:

```
# service syslogd restart
```

To test that log messages are being sent across the network, use logger(1) on the client to send a message to syslogd:

```
# logger "Test message from logclient"
```

This message should now exist both in /var/log/messages on the client and /var/log/logclient.log on the log server.

Debugging Log Servers

If no messages are being received on the log server, the cause is most likely a network connectivity issue, a hostname resolution issue, or a typo in a configuration file. To isolate the cause, ensure that both the logging server and the logging client are able to `ping` each other using the hostname specified in their /etc/rc.conf. If this fails, check the network cabling, the firewall ruleset, and the hostname entries in the DNS server or /etc/hosts on both the logging server and clients. Repeat until the `ping` is successful from both hosts.

If the `ping` succeeds on both hosts but log messages are still not being received, temporarily increase logging verbosity to narrow down the configuration issue. In the following example, /var/log/logclient.log on the logging server is empty and /var/log/messages on the logging client does not indicate a reason for the failure.

To increase debugging output, edit the `syslogd_flags` entry on the logging server and issue a restart:

```
sysrc syslogd_flags="-d -a logclient.example.com -v -v"
```

```
# service syslogd restart
```

Debugging data similar to the following will flash on the console immediately after the restart:

```
logmsg: pri 56, flags 4, from logserv.example.com, msg syslogd: restart
syslogd: restarted
logmsg: pri 6, flags 4, from logserv.example.com, msg syslogd: kernel boot file is
/boot/kernel/kernel
Logging to FILE /var/log/messages
syslogd: kernel boot file is /boot/kernel/kernel
cvthname(192.168.1.10)
validate: dgram from IP 192.168.1.10, port 514, name logclient.example.com;
rejected in rule 0 due to name mismatch.
```

In this example, the log messages are being rejected due to a typo which results in a hostname mismatch. The client's hostname should be `logclient`, not `logclien`. Fix the typo, issue a restart, and verify the results:

```
# service syslogd restart
```

The output should be similar to the following:

```
logmsg: pri 56, flags 4, from logserv.example.com, msg syslogd: restart
syslogd: restarted
logmsg: pri 6, flags 4, from logserv.example.com, msg syslogd: kernel boot file is
/boot/kernel/kernel
syslogd: kernel boot file is /boot/kernel/kernel
logmsg: pri 166, flags 17, from logserv.example.com,
msg Dec 10 20:55:02 <syslog.err> logserv.example.com syslogd: exiting on signal 2
cvthname(192.168.1.10)
validate: dgram from IP 192.168.1.10, port 514, name logclient.example.com;
accepted in rule 0.
logmsg: pri 15, flags 0, from logclient.example.com, msg Dec 11 02:01:28 trhodes: Test
message 2
Logging to FILE /var/log/logclient.log
Logging to FILE /var/log/messages
```

At this point, the messages are being properly received and placed in the correct file.

Security Considerations

As with any network service, security requirements should be considered before implementing a logging server. Log files may contain sensitive data about services enabled on the local host, user accounts, and configuration data. Network data sent from the client to the server will not be encrypted or password protected. If a need for encryption exists, consider using security/stunnel, which will transmit the logging data over an encrypted tunnel.

Local security is also an issue. Log files are not encrypted during use or after log rotation. Local users may access log files to gain additional insight into system configuration. Setting proper permissions on log files is critical. The built-in log rotator, newsyslog, supports setting permissions on newly created and rotated log files. Setting log files to mode `600` should prevent unwanted access by local users. Refer to newsyslog.conf(5) for additional information.

14.6. Power and Resource Management

It is important to utilize hardware resources in an efficient manner. Power and resource management allows the operating system to monitor system limits and to possibly run some actions triggered by events related to those limits.

14.6.1. ACPI configuration

On FreeBSD the management of these resources is managed by the acpi(4) kernel device.

> In FreeBSD the acpi(4) driver is loaded by default at system boot.
>
> This driver **cannot be unloaded after boot** because the system bus uses it for various hardware interactions.

In addition to acpi(4), FreeBSD has several dedicated kernel modules for various ACPI vendor subsystems. These modules will add some extra functionality like fan speed, keyboard backlit or screen brightness.

The list can be obtained by running the following command:

```
% ls /boot/kernel | grep acpi
```

The output should be similar to the following:

```
acpi_asus.ko
acpi_asus_wmi.ko
acpi_dock.ko
acpi_fujitsu.ko
acpi_hp.ko
acpi_ibm.ko
acpi_panasonic.ko
acpi_sony.ko
acpi_toshiba.ko
acpi_video.ko
acpi_wmi.ko
sdhci_acpi.ko
```

```
uacpi.ko
```

In the event that, for example, an IBM/Lenovo laptop is used, it will be necessary to load the module acpi_ibm(4) by executing the following command:

```
# kldload acpi_ibm
```

And add this line to /boot/loader.conf to load it at boot:

```
acpi_ibm_load="YES"
```

An alternative to the acpi_video(4) module is the backlight(9) driver. It provides a generic way for handling a panel backlight. The default GENERIC kernel includes this driver. The backlight(8) utility can be used to query and adjust the brightness of the panel backlight. In this example the brightness is decreased by 10%:

```
% backlight decr 10
```

14.6.2. CPU Power Management

CPU is the most consuming part of the system. Knowing how to improve CPU efficiency is a fundamental part of our system in order to save energy.

In order to make proper use of the machine’s resources in a correct way, FreeBSD supports technologies such as Intel Turbo Boost, AMD Turbo Core, Intel Speed Shift among others through the use of powerd(8) and cpufreq(4).

The first step will be to obtain the CPU information by executing the following command:

```
% sysctl dev.cpu.0 ①
```

① In this case the `0` digit represents the first core of the CPU.

The output should be similar to the following:

```
dev.cpu.0.cx_method: C1/mwait/hwc C2/mwait/hwc C3/mwait/hwc/bma
dev.cpu.0.cx_usage_counters: 3507294 0 0
dev.cpu.0.cx_usage: 100.00% 0.00% 0.00% last 3804us
dev.cpu.0.cx_lowest: C3 ①
dev.cpu.0.cx_supported: C1/1/1 C2/2/1 C3/3/57 ②
dev.cpu.0.freq_levels: 2267/35000 2266/35000 1600/15000 800/12000 ③
dev.cpu.0.freq: 1600 ④
dev.cpu.0.temperature: 40.0C ⑤
dev.cpu.0.coretemp.throttle_log: 0
dev.cpu.0.coretemp.tjmax: 105.0C
dev.cpu.0.coretemp.resolution: 1
dev.cpu.0.coretemp.delta: 65
dev.cpu.0.%parent: acpi0
dev.cpu.0.%pnpinfo: _HID=none _UID=0 _CID=none
dev.cpu.0.%location: handle=\_PR_.CPU0
dev.cpu.0.%driver: cpu
dev.cpu.0.%desc: ACPI CPU
```

① Lowest Cx state to use for idling the CPU.

② CPU supported Cx states.

③ Currently available levels for the CPU (frequency/power usage).

④ Current active CPU frequency in MHz.

⑤ Current temperature of the CPU.

If the temperature information is not displayed, load the coretemp(4) module. In case of using an AMD CPU, load the amdtemp(4) module.

Once the CPU information is available the easiest way to configure power saving is to let powerd(8) take over.

Enable powerd(8) service in /etc/rc.conf to start at system boot:

```
# sysrc powerd_enable=YES
```

It will also be necessary to indicate certain parameters to powerd(8) to tell it how to manage the state of the CPU executing the following command:

```
# sysrc powerd_flags="-a hiadaptive -i 25 -r 85 -N"
```

1. `-a`: Selects the mode to use while on AC power.
2. `hiadaptive`: Operation mode. More info at powerd(8).
3. `-i`: Specifies the CPU load percent level when adaptive mode should begin to degrade performance to save power.
4. `-r`: Specifies the CPU load percent level where adaptive mode should consider the CPU running and increase performance.
5. `-N`: Treat "nice" time as idle for the purpose of load calculation; i.e., do not increase the CPU frequency if the CPU is only busy with "nice" processes.

And then enable the service executing the following command:

```
# service powerd start
```

14.6.3. CPU Frequency Control

FreeBSD includes a generic cpufreq(4) driver to allow the administrator, or software such as powerd(8) and sysutils/powerdxx, to manage the frequency of the CPU to achieve the desired balance between performance and economy. A lower setting will save power while reducing the heat generated by the CPU. A higher setting will increase performance at the cost of using additional power and generating more heat.

14.6.4. Intel® Enhanced Speed Step™

The Intel® Enhanced Speed Step™ driver, est(4), replaces the generic cpufreq(4) driver for CPUs that provide this feature. The CPU frequency can be statically adjusted using sysctl(8), or with the `/etc/rc.d/power_profile` startup script. Additional software, such as powerd(8) or sysutils/powerdxx, can be used to automatically adjust the CPU frequency based on processor utilization.

Each supported frequency, along with its expected power consumption, can be listed by examining the sysctl(3) tree:

```
# sysctl dev.cpufreq.0.freq_driver dev.cpu.0.freq_levels dev.cpu.0.freq
```

The output should be similar to the following:

```
dev.cpufreq.0.freq_driver: est0
dev.cpu.0.freq_levels: 3001/53000 3000/53000 2900/50301 2700/46082 2600/43525
2400/39557 2300/37137 2100/33398 2000/31112 1800/27610 1700/25455 1500/22171
1400/20144 1200/17084 1100/15181 900/12329 800/10550
dev.cpu.0.freq: 800
```

A frequency 1 MHz higher than the maximum frequency of the CPU indicates the Intel® Turbo Boost™ feature.

14.6.5. Intel Speed Shift™

Users running newer Intel® CPUs may find some differences in dynamic frequency control when upgrading to FreeBSD 13. A new driver for the Intel® Speed Shift™ feature set, available on certain SKUs, exposes the ability for the hardware to dynamically vary the core frequencies, including on a per core basis. FreeBSD 13 comes with the hwpstate_intel(4) driver to automatically enable Speed Shift™ control on equipped CPUs, replacing the older Enhanced Speed Step™ est(4) driver. The sysctl(8) `dev.cpufreq.%d.freq_driver` will indicate if the system is using Speed Shift.

To determine which frequency control driver is being used, examining the `dev.cpufreq.0.freq_driver` oid.

```
# sysctl dev.cpufreq.0.freq_driver
```

The output should be similar to the following:

```
dev.cpufreq.0.freq_driver: hwpstate_intel0
```

This indicates that the new hwpstate_intel(4) driver is in use. On such systems, the oid `dev.cpu.%d.freq_levels` will show only the maximum CPU frequency, and will indicate a power consumption level of `-1`.

The current CPU frequency can be determined by examining the `dev.cpu.%d.freq` oid.

```
# sysctl dev.cpu.0.freq_levels dev.cpu.0.freq
```

The output should be similar to the following:

```
dev.cpu.0.freq_levels: 3696/-1
dev.cpu.0.freq: 898
```

For more information, including on how to balance performance and energy use, and on how to disable this driver, refer to the man page hwpstate_intel(4).

Users accustomed to using powerd(8) or sysutils/powerdxx will find these utilities have been superseded by the hwpstate_intel(4) driver and no longer work as expected.

14.6.6. Graphics Card Power Management

Graphics cards have become a fundamental part of computing in recent years. Some graphics cards may have excessive power consumption. FreeBSD allows certain configurations to improve power consumption.

In case of using a Intel® graphics card with the graphics/drm-kmod driver these options can be added to /boot/loader.conf:

```
compat.linuxkpi.fastboot=1 ①
compat.linuxkpi.enable_dc=2 ②
compat.linuxkpi.enable_fbc=1 ③
```

① Try to skip unnecessary mode sets at boot time.

② Enable power-saving display C-states.

③ Enable frame buffer compression for power savings

14.6.7. Suspend/Resume

The suspend/resume function allows the machine to be kept in a state in which there is no a big energy consumption and allows the system to be resumed without having to lose the state of the running programs.

In order for the suspend/resume functionality to work correctly the graphics drivers must be loaded on the system. In non-KMS-supported graphics cards sc(4) must be used not to break the suspend/resume functionality.

More information about which driver to use and how to configure it can be found at the The X Window System chapter.

acpi(4) supports the next list of sleep states:

Table 27. Supported Sleep States

S1	**Quick suspend to RAM. The CPU enters a lower power state, but most peripherals are left running.**
S2	Lower power state than S1, but with the same basic characteristics. Not supported by many systems.
S3 (Sleep mode)	Suspend to RAM. Most devices are powered off, and the system stops running except for memory refresh.
S4 (Hibernation)	Suspend to disk. All devices are powered off, and the system stops running. When resuming, the system starts as if from a cold power on. **Not yet supported by FreeBSD**.
S5	System shuts down cleanly and powers off.

Configuring Suspend/Resume

The first step will be to know which type of sleep states supports the hardware we are using executing the following command:

```
% sysctl hw.acpi.supported_sleep_state
```

The output should be similar to the following:

```
hw.acpi.supported_sleep_state: S3 S4 S5
```

As stated above FreeBSD does **not** yet support the `S4` state.

acpiconf(8) can be used to check if the `S3` state works correctly by running the following command, if it succeeds, the screen should go black and the machine will turn off:

```
# acpiconf -s 3
```

In the vast majority of cases the Suspend/Resume functionality wants to be used on a laptop.

FreeBSD can be configured to enter the `S3` state when closing the lid by adding the following line to the /etc/sysctl.conf file.

```
hw.acpi.lid_switch_state=S3
```

Troubleshooting in Suspend/Resume

A lot of effort has been made to make the Suspend and Resume functions work properly and in the best way on FreeBSD. But currently the Suspend and Resume functions only work properly on some specific laptops.

Some checks can be done in case it doesn't work properly.

In some cases it is enough to turn off the bluetooth. In others it is enough loading the correct driver for the graphics card, etc.

In case it doesn't work correctly, some tips can be found on the FreeBSD Wiki in the section Suspend/Resume.

14.7. Adding Swap Space

Sometimes a FreeBSD system requires more swap space. This section describes two methods to increase swap space: adding swap to an existing partition or new hard drive, and creating a swap file on an existing file system.

For information on how to encrypt swap space, which options exist, and why it should be done, refer to "Encrypting Swap".

14.7.1. Swap on a New Hard Drive or Existing Partition

Adding a new drive for swap gives better performance than using a partition on an existing drive. Setting up partitions and drives is explained in Adding Disks while Designing the Partition Layout discusses partition layouts and swap partition size considerations.

> ⚠ It is possible to use any partition not currently mounted, even if it already contains data. Using `swapon` on a partition that contains data will overwrite and destroy that data. Make sure that the partition to be added as swap is really the intended partition before running `swapon`.

swapon(8) can be used to add a swap partition to the system executing the following command:

```
# swapon /dev/ada1p2
```

To automatically add this swap partition on boot, add an entry to /etc/fstab:

```
/dev/ada1p2 none swap sw 0 0
```

See fstab(5) for an explanation of the entries in /etc/fstab.

14.7.2. Creating a Swap File

These examples create a 512M swap file called /usr/swap0.

> Swap files on ZFS file systems are strongly discouraged, as swapping can lead to system hangs.

The first step is to create the swap file:

```
# dd if=/dev/zero of=/usr/swap0 bs=1m count=512
```

The second step is to put the proper permissions on the new file:

```
# chmod 0600 /usr/swap0
```

The third step is to inform the system about the swap file by adding a line to /etc/fstab:

```
md none swap sw,file=/usr/swap0,late 0 0
```

Swap space will be added on system startup. To add swap space immediately, use swapon(8):

```
# swapon -aL
```

Chapter 15. The FreeBSD Booting Process

15.1. Synopsis

The process of starting a computer and loading the operating system is referred to as "the bootstrap process", or "booting". FreeBSD’s boot process provides a great deal of flexibility in customizing what happens when the system starts, including the ability to select from different operating systems installed on the same computer, different versions of the same operating system, or a different installed kernel.

This chapter details the configuration options that can be set. It demonstrates how to customize the FreeBSD boot process, including everything that happens until the FreeBSD kernel has started, probed for devices, and started init(8). This occurs when the text color of the boot messages changes from bright white to grey.

Read this chapter to learn:

- The components of the FreeBSD bootstrap system and how they interact.
- The options that can be passed to the components in the FreeBSD bootstrap in order to control the boot process.
- The basics of setting device hints.
- How to boot into single- and multi-user mode and how to properly shut down a FreeBSD system.

> This chapter only describes the boot process for FreeBSD running on x86 and amd64 systems.

15.2. FreeBSD Boot Process

Turning on a computer and starting the operating system poses an interesting dilemma. By definition, the computer does not know how to do anything until the operating system is started. This includes running programs from the disk. If the computer can not run a program from the disk without the operating system, and the operating system programs are on the disk, how is the operating system started?

This problem parallels one in the book The Adventures of Baron Munchausen. A character had fallen part way down a manhole, and pulled himself out by grabbing his bootstraps and lifting. In the early days of computing, the term *bootstrap* was applied to the mechanism used to load the operating system. It has since become shortened to "booting".

On x86 hardware, the Basic Input/Output System (BIOS) is responsible for loading the operating system. The BIOS looks on the hard disk for the Master Boot Record (MBR), which must be located in a specific place on the disk. The BIOS has enough knowledge to load and run the MBR, and assumes that the MBR can then carry out the rest of the tasks involved in loading the operating system, possibly with the help of the BIOS.

> FreeBSD provides for booting from both the older MBR standard, and the newer

GUID Partition Table (GPT). GPT partitioning is often found on computers with the Unified Extensible Firmware Interface (UEFI). However, FreeBSD can boot from GPT partitions even on machines with only a legacy BIOS with gptboot(8). Work is under way to provide direct UEFI booting.

The code within the MBR is typically referred to as a *boot manager*, especially when it interacts with the user. The boot manager usually has more code in the first track of the disk or within the file system. Examples of boot managers include the standard FreeBSD boot manager boot0, also called Boot Easy, and GNU GRUB, which is used by many Linux® distributions.

Users of GRUB should refer to GNU-provided documentation.

If only one operating system is installed, the MBR searches for the first bootable (active) slice on the disk, and then runs the code on that slice to load the remainder of the operating system. When multiple operating systems are present, a different boot manager can be installed to display a list of operating systems so the user can select one to boot.

The remainder of the FreeBSD bootstrap system is divided into three stages. The first stage knows just enough to get the computer into a specific state and run the second stage. The second stage can do a little bit more, before running the third stage. The third stage finishes the task of loading the operating system. The work is split into three stages because the MBR puts limits on the size of the programs that can be run at stages one and two. Chaining the tasks together allows FreeBSD to provide a more flexible loader.

The kernel is then started and begins to probe for devices and initialize them for use. Once the kernel boot process is finished, the kernel passes control to the user process init(8), which makes sure the disks are in a usable state, starts the user-level resource configuration which mounts file systems, sets up network cards to communicate on the network, and starts the processes which have been configured to run at startup.

This section describes these stages in more detail and demonstrates how to interact with the FreeBSD boot process.

15.2.1. The Boot Manager

The boot manager code in the MBR is sometimes referred to as *stage zero* of the boot process. By default, FreeBSD uses the boot0 boot manager.

The MBR installed by the FreeBSD installer is based on /boot/boot0. The size and capability of boot0 is restricted to 446 bytes due to the slice table and `0x55AA` identifier at the end of the MBR. If boot0 and multiple operating systems are installed, a message similar to this example will be displayed at boot time:

Example 23. boot0 Screenshot

```
F1 Win
F2 FreeBSD
```

```
Default: F2
```

Other operating systems will overwrite an existing MBR if they are installed after FreeBSD. If this happens, or to replace the existing MBR with the FreeBSD MBR, use the following command:

```
# fdisk -B -b /boot/boot0 device
```

where *device* is the boot disk, such as ad0 for the first IDE disk, ad2 for the first IDE disk on a second IDE controller, or da0 for the first SCSI disk. To create a custom configuration of the MBR, refer to boot0cfg(8).

15.2.2. Stage One and Stage Two

Conceptually, the first and second stages are part of the same program on the same area of the disk. Due to space constraints, they have been split into two, but are always installed together. They are copied from the combined /boot/boot by the FreeBSD installer or `bsdlabel`.

These two stages are located outside file systems, in the first track of the boot slice, starting with the first sector. This is where boot0, or any other boot manager, expects to find a program to run which will continue the boot process.

The first stage, boot1, is very simple, since it can only be 512 bytes in size. It knows just enough about the FreeBSD *bsdlabel*, which stores information about the slice, to find and execute boot2.

Stage two, boot2, is slightly more sophisticated, and understands the FreeBSD file system enough to find files. It can provide a simple interface to choose the kernel or loader to run. It runs loader, which is much more sophisticated and provides a boot configuration file. If the boot process is interrupted at stage two, the following interactive screen is displayed:

Example 24. boot2 Screenshot

```
>> FreeBSD/i386 BOOT
Default: 0:ad(0,a)/boot/loader
boot:
```

To replace the installed boot1 and boot2, use `bsdlabel`, where *diskslice* is the disk and slice to boot from, such as ad0s1 for the first slice on the first IDE disk:

```
# bsdlabel -B diskslice
```

> If just the disk name is used, such as ad0, `bsdlabel` will create the disk in "dangerously dedicated mode", without slices. This is probably not the desired action, so double check the *diskslice* before pressing `Return`.

15.2.3. Stage Three

The loader is the final stage of the three-stage bootstrap process. It is located on the file system, usually as /boot/loader.

The loader is intended as an interactive method for configuration, using a built-in command set, backed up by a more powerful interpreter which has a more complex command set.

During initialization, loader will probe for a console and for disks, and figure out which disk it is booting from. It will set variables accordingly, and an interpreter is started where user commands can be passed from a script or interactively.

The loader will then read /boot/loader.rc, which by default reads in /boot/defaults/loader.conf which sets reasonable defaults for variables and reads /boot/loader.conf for local changes to those variables. loader.rc then acts on these variables, loading whichever modules and kernel are selected.

Finally, by default, loader issues a 10 second wait for key presses, and boots the kernel if it is not interrupted. If interrupted, the user is presented with a prompt which understands the command set, where the user may adjust variables, unload all modules, load modules, and then finally boot or reboot. Loader Built-In Commands lists the most commonly used loader commands. For a complete discussion of all available commands, refer to loader(8).

Table 28. Loader Built-In Commands

Variable	Description
autoboot *seconds*	Proceeds to boot the kernel if not interrupted within the time span given, in seconds. It displays a countdown, and the default time span is 10 seconds.
boot [`-options`] [`kernelname`]	Immediately proceeds to boot the kernel, with any specified options or kernel name. Providing a kernel name on the command-line is only applicable after an `unload` has been issued. Otherwise, the previously-loaded kernel will be used. If *kernelname* is not provided with the full path, it will be searched under */boot/kernel* and */boot/modules*.
boot-conf	Goes through the same automatic configuration of modules based on specified variables, most commonly `kernel`. This only makes sense if `unload` is used first, before changing some variables.
help [`topic`]	Shows help messages read from /boot/loader.help. If the topic given is `index`, the list of available topics is displayed.
include `filename` …	Reads the specified file and interprets it line by line. An error immediately stops the `include`.
load [-t `type`] `filename`	Loads the kernel, kernel module, or file of the type given, with the specified filename. Any arguments after *filename* are passed to the file. If *filename* is not provided with the full path, it will be searched under */boot/kernel* and */boot/modules*.
ls [-l] [`path`]	Displays a listing of files in the given path, or the root directory, if the path is not specified. If `-l` is specified, file sizes will also be shown.

Variable	Description
lsdev [`-v`]	Lists all of the devices from which it may be possible to load modules. If `-v` is specified, more details are printed.
lsmod [`-v`]	Displays loaded modules. If `-v` is specified, more details are shown.
more `filename`	Displays the files specified, with a pause at each `LINES` displayed.
reboot	Immediately reboots the system.
set `variable`, set `variable=value`	Sets the specified environment variables.
unload	Removes all loaded modules.

Here are some practical examples of loader usage. To boot the usual kernel in single-user mode:

```
boot -s
```

To unload the usual kernel and modules and then load the previous or another, specified kernel:

```
unload
load /path/to/kernelfile
```

Use the full path /boot/GENERIC/kernel to refer to the default kernel that comes with an installation, or /boot/kernel.old/kernel, to refer to the previously installed kernel before a system upgrade or before configuring a custom kernel.

Use the following to load the usual modules with another kernel. Note that in this case it is not necessary to provide the full path, only the file name:

```
unload
set kernel="mykernel"
boot-conf
```

To load an automated kernel configuration script:

```
load -t userconfig_script /boot/kernel.conf
```

15.2.4. Last Stage

Once the kernel is loaded by either loader or by boot2, which bypasses loader, it examines any boot flags and adjusts its behavior as necessary. Kernel Interaction During Boot lists the commonly used boot flags. Refer to boot(8) for more information on the other boot flags.

Table 29. Kernel Interaction During Boot

Option	Description
`-a`	During kernel initialization, ask for the device to mount as the root file system.
`-C`	Boot the root file system from a CDROM.
`-s`	Boot into single-user mode.
`-v`	Be more verbose during kernel startup.

Once the kernel has finished booting, it passes control to the user process init(8), which is located at /sbin/init, or the program path specified in the `init_path` variable in `loader`. This is the last stage of the boot process.

The boot sequence makes sure that the file systems available on the system are consistent. If a UFS file system is not, and `fsck` cannot fix the inconsistencies, init drops the system into single-user mode so that the system administrator can resolve the problem directly. Otherwise, the system boots into multi-user mode.

Single-User Mode

A user can specify this mode by booting with `-s` or by setting the `boot_single` variable in loader. It can also be reached by running `shutdown now` from multi-user mode. Single-user mode begins with this message:

```
Enter full pathname of shell or RETURN for /bin/sh:
```

If the user presses `Enter`, the system will enter the default POSIX® shell. To specify a different shell, input the full path to the shell.

Single-user mode is usually used to repair a system that will not boot due to an inconsistent file system or an error in a boot configuration file. It can also be used to reset the `root` password when it is unknown. These actions are possible as the single-user mode prompt gives full, local access to the system and its configuration files. There is no networking in this mode.

While single-user mode is useful for repairing a system, it poses a security risk unless the system is in a physically secure location. By default, any user who can gain physical access to a system will have full control of that system after booting into single-user mode.

If the system `console` is changed to `insecure` in /etc/ttys, the system will first prompt for the `root` password before initiating single-user mode. This adds a measure of security while removing the ability to reset the `root` password when it is unknown.

Example 25. Configuring an Insecure Console in /etc/ttys

```
# name  getty                           type    status          comments
#
# If console is marked "insecure", then init will ask for the root password
# when going to single-user mode.
```

```
console none                                unknown off insecure
```

An `insecure` console means that physical security to the console is considered to be insecure, so only someone who knows the `root` password may use single-user mode.

Multi-User Mode

If init finds the file systems to be in order, or once the user has finished their commands in single-user mode and has typed `exit` to leave single-user mode, the system enters multi-user mode, in which it starts the resource configuration of the system.

The resource configuration system reads in configuration defaults from /etc/defaults/rc.conf and system-specific details from /etc/rc.conf. It then proceeds to mount the system file systems listed in /etc/fstab. It starts up networking services, miscellaneous system daemons, then the startup scripts of locally installed packages.

To learn more about the resource configuration system, refer to rc(8) and examine the scripts located in /etc/rc.d.

15.3. Device Hints

During initial system startup, the boot loader(8) reads device.hints(5). This file stores kernel boot information known as variables, sometimes referred to as "device hints". These "device hints" are used by device drivers for device configuration.

Device hints may also be specified at the Stage 3 boot loader prompt, as demonstrated in Stage Three. Variables can be added using `set`, removed with `unset`, and viewed `show`. Variables set in /boot/device.hints can also be overridden. Device hints entered at the boot loader are not permanent and will not be applied on the next reboot.

Once the system is booted, kenv(1) can be used to dump all of the variables.

The syntax for /boot/device.hints is one variable per line, using the hash "#" as comment markers. Lines are constructed as follows:

```
hint.driver.unit.keyword="value"
```

The syntax for the Stage 3 boot loader is:

```
set hint.driver.unit.keyword=value
```

where `driver` is the device driver name, `unit` is the device driver unit number, and `keyword` is the hint keyword. The keyword may consist of the following options:

- `at`: specifies the bus which the device is attached to.
- `port`: specifies the start address of the I/O to be used.

- `irq`: specifies the interrupt request number to be used.
- `drq`: specifies the DMA channel number.
- `maddr`: specifies the physical memory address occupied by the device.
- `flags`: sets various flag bits for the device.
- `disabled`: if set to `1` the device is disabled.

Since device drivers may accept or require more hints not listed here, viewing a driver's manual page is recommended. For more information, refer to device.hints(5), kenv(1), loader.conf(5), and loader(8).

15.4. Shutdown Sequence

Upon controlled shutdown using shutdown(8), init(8) will attempt to run the script /etc/rc.shutdown, and then proceed to send all processes the `TERM` signal, and subsequently the `KILL` signal to any that do not terminate in a timely manner.

To power down a FreeBSD machine on architectures and systems that support power management, use `shutdown -p now` to turn the power off immediately. To reboot a FreeBSD system, use `shutdown -r now`. One must be `root` or a member of `operator` in order to run shutdown(8). One can also use halt(8) and reboot(8). Refer to their manual pages and to shutdown(8) for more information.

Modify group membership by referring to "Users and Basic Account Management".

Power management requires acpi(4) to be loaded as a module or statically compiled into a custom kernel.

Chapter 16. Security

16.1. Synopsis

Hundreds of standard practices have been authored about how to secure systems and networks, and as a user of FreeBSD, understanding how to protect against attacks and intruders is a must.

In this chapter, several fundamentals and techniques will be discussed. The FreeBSD system comes with multiple layers of security, and many more third party utilities may be added to enhance security.

This chapter covers:

- Basic FreeBSD system security concepts.
- The various crypt mechanisms available in FreeBSD.
- How to configure TCP Wrappers for use with inetd(8).
- How to set up Kerberos on FreeBSD.
- How to configure and use OpenSSH on FreeBSD.
- How to use OpenSSL on FreeBSD.
- How to use file system ACLs.
- How to use pkg to audit third party software packages installed from the Ports Collection.
- How to utilize FreeBSD security advisories.
- What Process Accounting is and how to enable it on FreeBSD.
- How to control user resources using login classes or the resource limits database.
- What is Capsicum and a basic example.

Certain topics due to their complexity are found in dedicated chapters such as Firewalls, Mandatory Access Control and articles like VPN over IPsec.

16.2. Introduction

Security is everyone’s responsibility. A weak entry point in any system could allow intruders to gain access to critical information and cause havoc on an entire network. One of the core principles of information security is the CIA triad, which stands for the Confidentiality, Integrity, and Availability of information systems.

The CIA triad is a bedrock concept of computer security as customers and users expect their data to be protected. For example, a customer expects that their credit card information is securely stored (confidentiality), that their orders are not changed behind the scenes (integrity), and that they have access to their order information at all times (availability).

To provide CIA, security professionals apply a defense in depth strategy. The idea of defense in depth is to add several layers of security to prevent one single layer failing and the entire security system collapsing. For example, a system administrator cannot simply turn on a firewall and consider the network or system secure. One must also audit accounts, check the integrity of binaries, and ensure malicious tools are not installed. To implement an effective security strategy, one must understand threats and how to defend against them.

What is a threat as it pertains to computer security? Threats are not limited to remote attackers who attempt to access a system without permission from a remote location. Threats also include employees, malicious software, unauthorized network devices, natural disasters, security vulnerabilities, and even competing corporations.

Systems and networks can be accessed without permission, sometimes by accident, or by remote attackers, and in some cases, via corporate espionage or former employees. As a user, it is important to prepare for and admit when a mistake has led to a security breach and report possible issues to the security team. As an administrator, it is important to know of the threats and be prepared to mitigate them.

When applying security to systems, it is recommended to start by securing the basic accounts and system configuration, and then to secure the network layer so that it adheres to the system policy and the organization's security procedures. Many organizations already have a security policy that covers the configuration of technology devices. The policy should include the security configuration of workstations, desktops, mobile devices, phones, production servers, and development servers. In many cases, standard operating procedures (SOPs) already exist. When in doubt, ask the security team.

16.3. Securing Accounts

Maintaining secure accounts in FreeBSD is crucial for data confidentiality, system integrity, and privilege separation, as it prevents unauthorized access, malware, and data breaches while ensuring compliance and protecting an organization's reputation.

16.3.1. Preventing Logins

In securing a system, a good starting point is an audit of accounts. Disable any accounts that do not need login access.

Ensure that `root` has a strong password and that this password is not shared.

To deny login access to accounts, two methods exist.

The first is to lock the account, this example shows how to lock the `imani` account:

```
# pw lock imani
```

The second method is to prevent login access by changing the shell to /usr/sbin/nologin. The nologin(8) shell prevents the system from assigning a shell to the user when they attempt to login.

Only the superuser can change the shell for other users:

```
# chsh -s /usr/sbin/nologin imani
```

16.3.2. Password Hashes

Passwords are a necessary evil of technology. When they must be used, they should be complex and a powerful hash mechanism should be used to encrypt the version that is stored in the password database. FreeBSD supports several algorithms, including SHA256, SHA512 and Blowfish hash algorithms in its `crypt()` library, see crypt(3) for details.

The default of SHA512 should not be changed to a less secure hashing algorithm, but can be changed to the more secure Blowfish algorithm.

> Blowfish is not part of AES and is not considered compliant with any Federal Information Processing Standards (FIPS). Its use may not be permitted in some environments.

To determine which hash algorithm is used to encrypt a user's password, the superuser can view the hash for the user in the FreeBSD password database. Each hash starts with a symbol which indicates the type of hash mechanism used to encrypt the password.

If DES is used, there is no beginning symbol. For MD5, the symbol is `$`. For SHA256 and SHA512, the symbol is `$6$`. For Blowfish, the symbol is `$2a$`. In this example, the password for `imani` is hashed using the default SHA512 algorithm as the hash starts with `$6$`. Note that the encrypted hash, not the password itself, is stored in the password database:

```
# grep imani /etc/master.passwd
```

The output should be similar to the following:

```
imani:$6$pzIjSvCAn.PBYQBA$PXpSeWPx3g5kscj3IMiM7tUEUSPmGexxta.8Lt9TGSi2lNQqYGKszsBPuGME
0:1001:1001::0:0:imani:/usr/home/imani:/bin/sh
```

The hash mechanism is set in the user's login class.

The following command can be run to check which hash mechanism is currently being used:

```
% grep passwd_format /etc/login.conf
```

The output should be similar to the following:

```
:passwd_format=sha512:\
```

For example, to change the algorithm to Blowfish, modify that line to look like this:

```
:passwd_format=blf:\
```

Then, cap_mkdb(1) must be executed to upgrade the login.conf database:

```
# cap_mkdb /etc/login.conf
```

Note that this change will not affect any existing password hashes. This means that all passwords should be re-hashed by asking users to run `passwd` in order to change their password.

16.3.3. Password Policy Enforcement

Enforcing a strong password policy for local accounts is a fundamental aspect of system security. In FreeBSD, password length, password strength, and password complexity can be implemented using built-in Pluggable Authentication Modules (PAM).

This section demonstrates how to configure the minimum and maximum password length and the enforcement of mixed characters using the pam_passwdqc(8) module. This module is enforced when a user changes their password.

To configure this module, become the superuser and uncomment the line containing `pam_passwdqc.so` in /etc/pam.d/passwd.

Then, edit that line to match the password policy:

```
password        requisite       pam_passwdqc.so
min=disabled,disabled,disabled,12,10 similar=deny retry=3 enforce=users
```

The explanation of the parameters can be found in pam_passwdqc(8).

Once this file is saved, a user changing their password will see a message similar to the following:

```
% passwd
```

The output should be similar to the following:

```
Changing local password for user
Old Password:

You can now choose the new password.
A valid password should be a mix of upper and lower case letters,
digits and other characters.  You can use a 12 character long
password with characters from at least 3 of these 4 classes, or
a 10 character long password containing characters from all the
classes.  Characters that form a common pattern are discarded by
the check.
```

```
Alternatively, if no one else can see your terminal now, you can
pick this as your password: "trait-useful&knob".
Enter new password:
```

If a password that does not match the policy is entered, it will be rejected with a warning and the user will have an opportunity to try again, up to the configured number of retries.

If the organization's policy requires passwords to expire, FreeBSD supports the `passwordtime` in the user's login class in /etc/login.conf

The `default` login class contains an example:

```
#       :passwordtime=90d:\
```

So, to set an expiry of 90 days for this login class, remove the comment symbol (#), save the edit, and execute the following command:

```
# cap_mkdb /etc/login.conf
```

To set the expiration on individual users, pass an expiration date or the number of days to expiry and a username to `pw`:

```
# pw usermod -p 30-apr-2025 -n user
```

As seen here, an expiration date is set in the form of day, month, and year. For more information, see pw(8).

16.3.4. Shared Administration with sudo

System administrators often need the ability to grant enhanced permissions to users so they may perform privileged tasks. The idea that team members are provided access to a FreeBSD system to perform their specific tasks opens up unique challenges to every administrator. These team members only need a subset of access beyond normal end user levels; however, they almost always tell management they are unable to perform their tasks without superuser access. Thankfully, there is no reason to provide such access to end users because tools exist to manage this exact requirement.

Even administrators should limit their privileges when not needed.

Up to this point, the security chapter has covered permitting access to authorized users and attempting to prevent unauthorized access. Another problem arises once authorized users have access to the system resources. In many cases, some users may need access to application startup scripts, or a team of administrators need to maintain the system. Traditionally, the standard users and groups, file permissions, and even the su(1) command would manage this access. And as applications required more access, as more users needed to use system resources, a better solution was required. The most used application is currently Sudo.

Sudo allows administrators to configure more rigid access to system commands and provide for some advanced logging features. As a tool, it is available from the Ports Collection as security/sudo or by use of the pkg(8) utility.

Execute the following command to install it:

```
# pkg install sudo
```

After the installation is complete, the installed `visudo` will open the configuration file with a text editor. Using `visudo` is highly recommended as it comes with a built in syntax checker to verify there are no errors before the file is saved.

The configuration file is made up of several small sections which allow for extensive configuration. In the following example, web application maintainer, user1, needs to start, stop, and restart the web application known as *webservice*. To grant this user permission to perform these tasks, add this line to the end of /usr/local/etc/sudoers:

```
user1   ALL=(ALL)       /usr/sbin/service webservice *
```

The user may now start *webservice* using this command:

```
% sudo /usr/sbin/service webservice start
```

While this configuration allows a single user access to the webservice service; however, in most organizations, there is an entire web team in charge of managing the service. A single line can also give access to an entire group. These steps will create a web group, add a user to this group, and allow all members of the group to manage the service:

```
# pw groupadd -g 6001 -n webteam
```

Using the same pw(8) command, the user is added to the webteam group:

```
# pw groupmod -m user1 -n webteam
```

Finally, this line in /usr/local/etc/sudoers allows any member of the webteam group to manage *webservice*:

```
%webteam   ALL=(ALL)       /usr/sbin/service webservice *
```

Unlike su(1), sudo(8) only requires the end user password. This avoids sharing passwords, which is a poor practice.

Users permitted to run applications with sudo(8) only enter their own passwords. This is more secure and gives better control than su(1), where the `root` password is entered and the user acquires all `root` permissions.

> Most organizations are moving or have moved toward a two factor authentication model. In these cases, the user may not have a password to enter.
>
> sudo(8) can be configured to permit two factor authentication model by using the `NOPASSWD` variable. Adding it to the configuration above will allow all members of the *webteam* group to manage the service without the password requirement:
>
> ```
> %webteam   ALL=(ALL)       NOPASSWD: /usr/sbin/service webservice *
> ```

16.3.5. Shared Administration with Doas

doas(1) is a command-line utility ported from OpenBSD. It serves as an alternative to the widely used sudo(8) command in Unix-like systems.

With doas, users can execute commands with elevated privileges, typically as the root user, while maintaining a simplified and security-conscious approach. Unlike sudo(8), doas emphasizes simplicity and minimalism, focusing on streamlined privilege delegation without an overwhelming array of configuration options.

Execute the following command to install it:

```
# pkg install doas
```

After the installation /usr/local/etc/doas.conf must be configured to grant access for users for specific commands, or roles.

The simplest entry could be the following, which grants the user `local_user` with `root` permissions without asking for its password when executing the doas command.

```
permit nopass local_user as root
```

After the installation and configuration of the `doas` utility, a command can now be executed with enhanced privileges, for example:

```
$ doas vi /etc/rc.conf
```

For more configuration examples, please read doas.conf(5).

16.4. Intrusion Detection System (IDS)

Verification of system files and binaries is important because it provides the system administration and security teams information about system changes. A software application that monitors the system for changes is called an Intrusion Detection System (IDS).

FreeBSD provides native support for a basic IDS system called mtree(8). While the nightly security emails will notify an administrator of changes, the information is stored locally and there is a chance that a malicious user could modify this information in order to hide their changes to the system. As such, it is recommended to create a separate set of binary signatures and store them on a read-only, root-owned directory or, preferably, on a removable USB disk or remote server.

It is also recommended to run `freebsd-update IDS` after each update.

16.4.1. Generating the Specification File

The built-in mtree(8) utility can be used to generate a specification of the contents of a directory. A seed, or a numeric constant, is used to generate the specification and is required to check that the specification has not changed. This makes it possible to determine if a file or binary has been modified. Since the seed value is unknown by an attacker, faking or checking the checksum values of files will be difficult to impossible.

It is recommended to create specifications for the directories which contain binaries and configuration files, as well as any directories containing sensitive data. Typically, specifications are created for /bin, /sbin, /usr/bin, /usr/sbin, /usr/local/bin, /etc, and /usr/local/etc.

The following example generates a set of `sha512` hashes, one for each system binary in /bin, and saves those values to a hidden file in user's home directory, /home/user/.bin_chksum_mtree:

```
# mtree -s 123456789 -c -K cksum,sha512 -p /bin > /home/user/.bin_chksum_mtree
```

The output should be similar to the following:

```
mtree: /bin checksum: 3427012225
```

The `123456789` value represents the seed, and should be chosen randomly. This value should be remembered, **but not shared**.

It is important to keep the seed value and the checksum output hidden from malicious users.

16.4.2. The Specification File Structure

The mtree format is a textual format that describes a collection of filesystem objects. Such files are typically used to create or verify directory hierarchies.

An mtree file consists of a series of lines, each providing information about a single filesystem object. Leading whitespace is always ignored.

The specification file created above will be used to explain the format and content:

```
#          user: root ①
#       machine: machinename ②
#          tree: /bin ③
#          date: Thu Aug  24 21:58:37 2023 ④

# .
/set type=file uid=0 gid=0 mode=0555 nlink=1 flags=uarch ⑤
.               type=dir mode=0755 nlink=2 time=1681388848.239523000 ⑥
    \133        nlink=2 size=12520 time=1685991378.688509000 \
                cksum=520880818 \

sha512=5c1374ce0e2ba1b3bc5a41b23f4bbdc1ec89ae82fa01237f376a5eeef41822e68f1d8f75ec46b7b
ceb65396c122a9d837d692740fdebdcc376a05275adbd3471
    cat         size=14600 time=1685991378.694601000 cksum=3672531848 \ ⑦

sha512=b30b96d155fdc4795432b523989a6581d71cdf69ba5f0ccb45d9b9e354b55a665899b16aee21982
fffe20c4680d11da4e3ed9611232a775c69f926e5385d53a2
    chflags     size=8920 time=1685991378.700385000 cksum=1629328991 \

sha512=289a088cbbcbeb436dd9c1f74521a89b66643976abda696b99b9cc1fbfe8b76107c5b54d4a6a9b6
5332386ada73fc1bbb10e43c4e3065fa2161e7be269eaf86a
    chio        size=20720 time=1685991378.706095000 cksum=1948751604 \

sha512=46f58277ff16c3495ea51e74129c73617f31351e250315c2b878a88708c2b8a7bb060e2dc8ff92f
606450dbc7dd2816da4853e465ec61ee411723e8bf52709ee
    chmod       size=9616 time=1685991378.712546000 cksum=4244658911 \

sha512=1769313ce08cba84ecdc2b9c07ef86d2b70a4206420dd71343867be7ab59659956f6f5a458c64e2
531a1c736277a8e419c633a31a8d3c7ccc43e99dd4d71d630
```

① User who created the specification.

② Machine’s hostname.

③ Directory path.

④ The Date and time when the specification was created.

⑤ `/set` special commands, defines some settings obtained from the files analyzed.

⑥ Refers to the parsed directory and indicates things like what type it is, its mode, the number of hard links, and the time in UNIX format since it was modified.

⑦ Refers to the file and shows the size, time and a list of hashes to verify the integrity.

16.4.3. Verify the Specification file

To verify that the binary signatures have not changed, compare the current contents of the directory to the previously generated specification, and save the results to a file.

This command requires the seed that was used to generate the original specification:

```
# mtree -s 123456789 -p /bin < /home/user/.bin_chksum_mtree >>
/home/user/.bin_chksum_output
```

This should produce the same checksum for /bin that was produced when the specification was created. If no changes have occurred to the binaries in this directory, the /home/user/.bin_chksum_output output file will be empty.

To simulate a change, change the date on /bin/cat using touch(1) and run the verification command again:

```
# touch /bin/cat
```

Run the verification command again:

```
# mtree -s 123456789 -p /bin < /home/user/.bin_chksum_mtree >>
/home/user/.bin_chksum_output
```

And then check the content of the output file:

```
# cat /root/.bin_chksum_output
```

The output should be similar to the following:

```
cat:    modification time (Fri Aug 25 13:30:17 2023, Fri Aug 25 13:34:20 2023)
```

This is just an example of what would be displayed when executing the command, to show the changes that would occur in the metadata.

16.5. Secure levels

securelevel is a security mechanism implemented in the kernel. When the securelevel is positive, the kernel restricts certain tasks; not even the superuser (root) is allowed to do them.

The securelevel mechanism limits the ability to:

- Unset certain file flags, such as `schg` (the system immutable flag).
- Write to kernel memory via /dev/mem and /dev/kmem.
- Load kernel modules.

- Alter firewall rules.

16.5.1. Secure Levels Definitions

The kernel runs with five different security levels. Any super-user process can raise the level, but no process can lower it.

The security definitions are:

-1

Permanently insecure mode - always run the system in insecure mode. This is the default initial value.

0

Insecure mode - immutable and append-only flags may be turned off. All devices may be read or written subject to their permissions.

1

Secure mode - the system immutable and system append-only flags may not be turned off; disks for mounted file systems, /dev/mem and /dev/kmem may not be opened for writing; /dev/io (if the platform has it) may not be opened at all; kernel modules (see kld(4)) may not be loaded or unloaded. The kernel debugger may not be entered using the debug.kdb.enter sysctl. A panic or trap cannot be forced using the debug.kdb.panic, debug.kdb.panic_str and other sysctl's.

2

Highly secure mode - same as secure mode, plus disks may not be opened for writing (except by mount(2)) whether mounted or not. This level precludes tampering with file systems by unmounting them, but also inhibits running newfs(8) while the system is multiuser.

3

Network secure mode - same as highly secure mode, plus IP packet filter rules (see ipfw(8), ipfirewall(4) and pfctl(8)) cannot be changed and dummynet(4) or pf(4) configuration cannot be adjusted.

In summary, the key difference between `Permanently Insecure Mode` and `Insecure Mode` in FreeBSD secure levels is the degree of security they provide. `Permanently Insecure Mode` completely lifts all security restrictions, while `Insecure Mode` relaxes some restrictions but still maintains a level of control and security.

16.5.2. Modify Secure Levels

In order to change the securelevel of the system it is necessary to activate `kern_securelevel_enable` by executing the following command:

```
# sysrc kern_securelevel_enable="YES"
```

And set the value of `kern_securelevel` to the desired security level:

```
# sysrc kern_securelevel=2
```

To check the status of the securelevel on a running system execute the following command:

```
# sysctl -n kern.securelevel
```

The output contains the current value of the securelevel. If it is greater than 0, at least some of the securelevel's protections are enabled.

16.6. File flags

File flags allow users to attach additional metadata or attributes to files and directories beyond basic permissions and ownership. These flags provide a way to control various behaviors and properties of files without needing to resort to creating special directories or using extended attributes.

File flags can be used to achieve different goals, such as preventing file deletion, making files append-only, synchronizing file updates, and more. Some commonly used file flags in FreeBSD include the "immutable" flag, which prevents modification or deletion of a file, and the "append-only" flag, which allows only data to be added to the end of a file but not modified or removed.

These flags can be managed using the chflags(1) command in FreeBSD, providing administrators and users with greater control over the behavior and characteristics of their files and directories. It is important to note that file flags are typically managed by root or users with appropriate privileges, as they can influence how files are accessed and manipulated. Some flags are available for the use of the file's owner, as described in chflags(1).

16.6.1. Work with File Flags

In this example, a file named ~/important.txt in user's home directory want to be protected against deletions.

Execute the following command to set the `schg` file flag:

```
# chflags schg ~/important.txt
```

When any user, including the `root` user, tries to delete the file, the system will display the message:

```
rm: important.txt: Operation not permitted
```

To delete the file, it will be necessary to delete the file flags of that file by executing the following command:

```
# chflags noschg ~/important.txt
```

A list of supported file flags and their functionality can be found in chflags(1).

16.7. OpenSSH

OpenSSH is a set of network connectivity tools used to provide secure access to remote machines. Additionally, TCP/IP connections can be tunneled or forwarded securely through SSH connections. OpenSSH encrypts all traffic to eliminate eavesdropping, connection hijacking, and other network-level attacks.

OpenSSH is maintained by the OpenBSD project and is installed by default in FreeBSD.

When data is sent over the network in an unencrypted form, network sniffers anywhere in between the client and server can steal user/password information or data transferred during the session. OpenSSH offers a variety of authentication and encryption methods to prevent this from happening.

More information about OpenSSH is available in the web page.

This section provides an overview of the built-in client utilities to securely access other systems and securely transfer files from a FreeBSD system. It then describes how to configure a SSH server on a FreeBSD system.

> As stated, this chapter will cover the base system version of OpenSSH. A version of OpenSSH is also available in the security/openssh-portable, which provides additional configuration options and is updated with new features more regularly.

16.7.1. Using the SSH Client Utilities

To log into a SSH server, use ssh(1) and specify a username that exists on that server and the IP address or hostname of the server. If this is the first time a connection has been made to the specified server, the user will be prompted to first verify the server's fingerprint:

```
# ssh user@example.com
```

The output should be similar to the following:

```
The authenticity of host 'example.com (10.0.0.1)' can't be established.
ECDSA key fingerprint is 25:cc:73:b5:b3:96:75:3d:56:19:49:d2:5c:1f:91:3b.
Are you sure you want to continue connecting (yes/no)? yes
Permanently added 'example.com' (ECDSA) to the list of known hosts.
Password for user@example.com: user_password
```

SSH utilizes a key fingerprint system to verify the authenticity of the server when the client connects. When the user accepts the key's fingerprint by typing `yes` when connecting for the first time, a copy of the key is saved to ~/.ssh/known_hosts in the user's home directory. Future attempts to login are verified against the saved key and ssh(1) will display an alert if the server's key does not match the saved key. If this occurs, the user should first verify why the key has changed before continuing with the connection.

How to perform this check is outside the scope of this chapter.

Use scp(1) to securely copy a file to or from a remote machine.

This example copies `COPYRIGHT` on the remote system to a file of the same name in the current directory of the local system:

```
# scp user@example.com:/COPYRIGHT COPYRIGHT
```

The output should be similar to the following:

```
Password for user@example.com: *******
COPYRIGHT            100% |*****************************|  4735
```

Since the fingerprint was already verified for this host, the server's key is automatically checked before prompting for the user's password.

The arguments passed to scp(1) are similar to cp(1). The file or files to copy is the first argument and the destination to copy to is the second. Since the file is fetched over the network, one or more of the file arguments takes the form `user@host:<path_to_remote_file>`. Be aware when copying directories recursively that scp(1) uses `-r`, whereas cp(1) uses `-R`.

To open an interactive session for copying files, use sftp(1).

Refer to sftp(1) for a list of available commands while in an sftp(1) session.

16.7.2. Key-based Authentication

Instead of using passwords, a client can be configured to connect to the remote machine using keys. For security reasons, this is the preferred method.

ssh-keygen(1) can be used to generate the authentication keys. To generate a public and private key pair, specify the type of key and follow the prompts. It is recommended to protect the keys with a memorable, but hard to guess passphrase.

```
% ssh-keygen -t rsa -b 4096
```

The output should be similar to the following:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
```

```
Created directory '/home/user/.ssh/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:54Xm9Uvtv6H4NOo6yjP/YCfODryvUU7yWHzMqeXwhq8 user@host.example.com
The key's randomart image is:
+---[RSA 2048]----+
|                 |
|                 |
|                 |
|        . o..    |
|       .S*+*o    |
|      . O=Oo . . |
|       = Oo= oo..|
|      .oB.* +.oo.|
|       =OE**.o..=|
+----[SHA256]-----+
```

The private key is stored in ~/.ssh/id_rsa and the public key is stored in ~/.ssh/id_rsa.pub. The *public* key must be copied to ~/.ssh/authorized_keys on the remote machine for key-based authentication to work.

Utilizing a passphrase for OpenSSH keys is a key security practice, providing an extra layer of protection against unauthorized access and enhancing overall cybersecurity.

In case of loss or theft, this adds another layer of security.

16.7.3. SSH Tunneling

OpenSSH has the ability to create a tunnel to encapsulate another protocol in an encrypted session.

The following command tells ssh(1) to create a tunnel:

```
% ssh -D 8080 user@example.com
```

This example uses the following options:

-D

Specifies a local "dynamic" application-level port forwarding.

user@foo.example.com

The login name to use on the specified remote SSH server.

An SSH tunnel works by creating a listen socket on `localhost` on the specified `localport`.

This method can be used to wrap any number of insecure TCP protocols such as SMTP, POP3, and FTP.

16.7.4. Enabling the SSH Server

In addition to providing built-in SSH client utilities, a FreeBSD system can be configured as an SSH server, accepting connections from other SSH clients.

> As stated, this chapter will cover the base system version of OpenSSH. Please **not** confuse with security/openssh-portable, the version of OpenSSH that ships with the FreeBSD ports.

In order to have the SSH Server enabled across reboots execute the following command:

```
# sysrc sshd_enable="YES"
```

Then execute the following command to enable the service:

```
# service sshd start
```

The first time sshd starts on a FreeBSD system, the system's host keys will be automatically created and the fingerprint will be displayed on the console. Provide users with the fingerprint so that they can verify it the first time they connect to the server.

Refer to sshd(8) for the list of available options when starting sshd and a complete discussion about authentication, the login process, and the various configuration files.

At this point, the sshd should be available to all users with a username and password on the system.

16.7.5. Configuring publickey auth method

Configuring OpenSSH to use public key authentication enhances security by leveraging asymmetric cryptography for authentication. This method eliminates password-related risks, such as weak passwords or interception during transmission, while thwarting various password-based attacks. However, it's vital to ensure the private keys are well-protected to prevent unauthorized access.

The first step will be to configure sshd(8) to use the required authentication method.

Edit /etc/ssh/sshd_config and uncomment the following configuration:

```
PubkeyAuthentication yes
```

Once the configuration is done, the users will have to send the system administrator their **public key** and these keys will be added in .ssh/authorized_keys. The process for generating the keys is described in Key-based Authentication.

Then restart the server executing the following command:

```
# service sshd reload
```

It is strongly recommended to follow the security improvements indicated in SSH Server Security Options.

16.7.6. SSH Server Security Options

While sshd is the most widely used remote administration facility for FreeBSD, brute force and drive by attacks are common to any system exposed to public networks.

Several additional parameters are available to prevent the success of these attacks and will be described in this section. All configurations will be done in /etc/ssh/sshd_config

Do not confuse /etc/ssh/sshd_config with /etc/ssh/ssh_config (note the extra `d` in the first filename). The first file configures the server and the second file configures the client. Refer to ssh_config(5) for a listing of the available client settings.

By default, authentication can be done with both pubkey and password. To allow **only** pubkey authentication, **which is strongly recommended**, change the variable:

```
PasswordAuthentication no
```

It is a good idea to limit which users can log into the SSH server and from where using the `AllowUsers` keyword in the OpenSSH server configuration file. For example, to only allow `user` to log in from `192.168.1.32`, add this line to /etc/ssh/sshd_config:

```
AllowUsers user@192.168.1.32
```

To allow `user` to log in from anywhere, list that user without specifying an IP address:

```
AllowUsers user
```

Multiple users should be listed on the same line, like so:

```
AllowUsers root@192.168.1.32 user
```

After making all the changes, and before restarting the service, it is recommended to verify that the configuration made is correct by executing the following command:

```
# sshd -t
```

If the configuration file is correct, no output will be shown. In case the configuration file is incorrect, it will show something like this:

```
/etc/ssh/sshd_config: line 3: Bad configuration option: sdadasdasdasads
/etc/ssh/sshd_config: terminating, 1 bad configuration options
```

After making the changes and checking that the configuration file is correct, tell sshd to reload its configuration file by running:

```
# service sshd reload
```

16.8. OpenSSL

OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and many cryptography routines.

The openssl program is a command line tool for using the various cryptography functions of OpenSSL's crypto library from the shell. It can be used for

- Creation and management of private keys, public keys and parameters
- Public key cryptographic operations
- Creation of X.509 certificates, CSRs and CRLs
- Calculation of Message Digests
- Encryption and Decryption with Ciphers
- SSL/TLS Client and Server Tests
- Handling of S/MIME signed or encrypted mail
- Time Stamp requests, generation and verification
- Benchmarking the crypto routines

For more information about OpenSSL, read the free OpenSSL Cookbook.

16.8.1. Generating Certificates

OpenSSL supports the generation of certificates both to be validated by a CA and for own use.

Run the command openssl(1) to generate a valid certificate for a CA with the following arguments. This command will create two files in the current directory. The certificate request, req.pem, can be sent to a CA which, will validate the entered credentials, sign the request, and return the signed certificate. The second file, cert.key, is the private key for the certificate and should be stored in a secure location. If this falls in the hands of others, it can be used to impersonate the user or the server.

Execute the following command to generate the certificate:

```
# openssl req -new -nodes -out req.pem -keyout cert.key -sha3-512 -newkey rsa:4096
```

The output should be similar to the following:

```
Generating a RSA private key
......................................................................................
............................................+++++
......................................+++++
writing new private key to 'cert.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valencian Community
Locality Name (eg, city) []:Valencia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Company
Organizational Unit Name (eg, section) []:Systems Administrator
Common Name (e.g. server FQDN or YOUR name) []:localhost.example.org
Email Address []:user@FreeBSD.org

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:123456789
An optional company name []:Another name
```

Alternately, if a signature from a CA is not required, a self-signed certificate can be created. This will create two new files in the current directory: a private key file cert.key, and the certificate itself, cert.crt. These should be placed in a directory, preferably under /etc/ssl/, which is readable only by `root`. Permissions of `0700` are appropriate for these files and can be set using `chmod`.

Execute the following command to generate the certificate:

```
# openssl req -new -x509 -days 365 -sha3-512 -keyout /etc/ssl/private/cert.key -out
/etc/ssl/certs/cert.crt
```

The output should be similar to the following:

```
Generating a RSA private key
........................................+++++
...........+++++
writing new private key to '/etc/ssl/private/cert.key'
Enter PEM pass phrase:
```

```
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:ES
State or Province Name (full name) [Some-State]:Valencian Community
Locality Name (eg, city) []:Valencia
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Company
Organizational Unit Name (eg, section) []:Systems Administrator
Common Name (e.g. server FQDN or YOUR name) []:localhost.example.org
Email Address []:user@FreeBSD.org
```

16.8.2. Configuring the FIPS Provider

With the import of OpenSSL 3 into the base system (on FreeBSD 14 and later), its new concept of provider modules was introduced in the system. Besides the default provider module built-in to the library, the *legacy* module implements the now optional deprecated cryptography algorithms, while the *fips* module restricts the OpenSSL implementation to the cryptography algorithms present in the FIPS set of standards. This part of OpenSSL receives particular care, including a list of relevant security issues, and is subject to the FIPS 140 validation process on a regular basis. The list of FIPS validated versions is also available. This allows users to ensure FIPS compliance in their use of OpenSSL.

Importantly, the fips_module(7) is protected by an additional security measure, preventing its use without passing an integrity check. This check can be setup by the local system administrator, allowing every user of OpenSSL 3 to load this module. When not configured correctly, the FIPS module is expected to fail as follows:

```
# echo test | openssl aes-128-cbc -a -provider fips -pbkdf2
```

The output should be similar to the following:

```
aes-128-cbc: unable to load provider fips
Hint: use -provider-path option or OPENSSL_MODULES environment variable.
00206124D94D0000:error:1C8000D5:Provider routines:SELF_TEST_post:missing config
data:crypto/openssl/providers/fips/self_test.c:275:
00206124D94D0000:error:1C8000E0:Provider routines:ossl_set_error_state:fips module
entering error state:crypto/openssl/providers/fips/self_test.c:373:
00206124D94D0000:error:1C8000D8:Provider routines:OSSL_provider_init_int:self test
post failure:crypto/openssl/providers/fips/fipsprov.c:707:
00206124D94D0000:error:078C0105:common libcrypto routines:provider_init:init
fail:crypto/openssl/crypto/provider_core.c:932:name=fips
```

The check can be configured through the creation of a file in /etc/ssl/fipsmodule.cnf, which will then be referenced in OpenSSL's main configuration file /etc/ssl/openssl.cnf. OpenSSL provides the openssl-fipsinstall(1) utility to help with this process, which can be used as follows:

```
# openssl fipsinstall -module /usr/lib/ossl-modules/fips.so -out
/etc/ssl/fipsmodule.cnf
```

The output should be similar to the following:

```
INSTALL PASSED
```

The /etc/ssl/openssl.cnf should then be modified, in order to:

- Include the /etc/ssl/fipsmodule.cnf file generated above,
- Expose the FIPS module for possible use,
- And explicitly activate the default module.

```
[...]
# For FIPS
# Optionally include a file that is generated by the OpenSSL fipsinstall
# application. This file contains configuration data required by the OpenSSL
# fips provider. It contains a named section e.g. [fips_sect] which is
# referenced from the [provider_sect] below.
# Refer to the OpenSSL security policy for more information.
.include /etc/ssl/fipsmodule.cnf

[...]

# List of providers to load
[provider_sect]
default = default_sect
# The fips section name should match the section name inside the
# included fipsmodule.cnf.
fips = fips_sect

# If no providers are activated explicitly, the default one is activated implicitly.
# See man 7 OSSL_PROVIDER-default for more details.
#
# If you add a section explicitly activating any other provider(s), you most
# probably need to explicitly activate the default provider, otherwise it
# becomes unavailable in openssl.  As a consequence applications depending on
# OpenSSL may not work correctly which could lead to significant system
# problems including inability to remotely access the system.
[default_sect]
activate = 1
```

With this done, it should be possible to confirm that the FIPS module is effectively available and working:

```
# echo test | openssl aes-128-cbc -a -provider fips -pbkdf2
```

The output should be similar to the following:

```
enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
U2FsdGVkX18idooW6e3LqWeeiKP76kufcOUClh57j8U=
```

This procedure has to be repeated every time the FIPS module is modified, e.g., after performing system updates, or after applying security fixes affecting OpenSSL in the base system.

16.9. Kerberos

Kerberos is a network authentication protocol which was originally created by the Massachusetts Institute of Technology (MIT) as a way to securely provide authentication across a potentially hostile network. The Kerberos protocol uses strong cryptography so that both a client and server can prove their identity without sending any unencrypted secrets over the network. Kerberos can be described as an identity-verifying proxy system and as a trusted third-party authentication system. After a user authenticates with Kerberos, their communications can be encrypted to assure privacy and data integrity.

The only function of Kerberos is to provide the secure authentication of users and servers on the network. It does not provide authorization or auditing functions. It is recommended that Kerberos be used with other security methods which provide authorization and audit services.

The current version of the protocol is version 5, described in RFC 4120. Several free implementations of this protocol are available, covering a wide range of operating systems. MIT continues to develop their Kerberos package. It is commonly used in the US as a cryptography product, and has historically been subject to US export regulations. In FreeBSD, MITKerberos is available as the security/krb5 package or port. The Heimdal Kerberos implementation was explicitly developed outside of the US to avoid export regulations. The Heimdal Kerberos distribution is included in the base FreeBSD installation, and another distribution with more configurable options is available as security/heimdal in the Ports Collection.

In Kerberos users and services are identified as "principals" which are contained within an administrative grouping, called a "realm". A typical user principal would be of the form `user@REALM` (realms are traditionally uppercase).

This section provides a guide on how to set up Kerberos using the Heimdal distribution included in FreeBSD.

For purposes of demonstrating a Kerberos installation, the name spaces will be as follows:

- The DNS domain (zone) will be `example.org`.

- The Kerberos realm will be `EXAMPLE.ORG`.

Use real domain names when setting up Kerberos, even if it will run internally. This avoids DNS problems and assures inter-operation with other Kerberos realms.

16.9.1. Setting up a Heimdal KDC

The Key Distribution Center (KDC) is the centralized authentication service that Kerberos provides, the "trusted third party" of the system. It is the computer that issues Kerberos tickets, which are used for clients to authenticate to servers. As the KDC is considered trusted by all other computers in the Kerberos realm, it has heightened security concerns. Direct access to the KDC should be limited.

While running a KDC requires few computing resources, a dedicated machine acting only as a KDC is recommended for security reasons.

To begin, install the security/heimdal package as follows:

```
# pkg install heimdal
```

Next, update /etc/rc.conf using `sysrc` as follows:

```
# sysrc kdc_enable=yes
# sysrc kadmind_enable=yes
```

Next, edit /etc/krb5.conf as follows:

```
[libdefaults]
    default_realm = EXAMPLE.ORG
[realms]
    EXAMPLE.ORG = {
    kdc = kerberos.example.org
    admin_server = kerberos.example.org
    }
[domain_realm]
    .example.org = EXAMPLE.ORG
```

In this example, the KDC will use the fully-qualified hostname `kerberos.example.org`. The hostname of the KDC must be resolvable in the DNS.

Kerberos can also use the DNS to locate KDCs, instead of a `[realms]` section in /etc/krb5.conf. For large organizations that have their own DNS servers, the above example could be trimmed to:

```
[libdefaults]
      default_realm = EXAMPLE.ORG
[domain_realm]
```

```
    .example.org = EXAMPLE.ORG
```

With the following lines being included in the `example.org` zone file:

```
_kerberos._udp      IN  SRV     01 00 88 kerberos.example.org.
_kerberos._tcp      IN  SRV     01 00 88 kerberos.example.org.
_kpasswd._udp       IN  SRV     01 00 464 kerberos.example.org.
_kerberos-adm._tcp  IN  SRV     01 00 749 kerberos.example.org.
_kerberos           IN  TXT     EXAMPLE.ORG
```

In order for clients to be able to find the Kerberos services, they *must* have either a fully configured /etc/krb5.conf or a minimally configured /etc/krb5.conf *and* a properly configured DNS server.

Next, create the Kerberos database which contains the keys of all principals (users and hosts) encrypted with a master password. It is not required to remember this password as it will be stored in /var/heimdal/m-key; it would be reasonable to use a 45-character random password for this purpose. To create the master key, run `kstash` and enter a password:

```
# kstash
```

The output should be similar to the following:

```
Master key: xxxxxxxxxxxxxxxxxxxxxxx
Verifying password - Master key: xxxxxxxxxxxxxxxxxxxxxxx
```

Once the master key has been created, the database should be initialized. The Kerberos administrative tool kadmin(8) can be used on the KDC in a mode that operates directly on the database, without using the kadmind(8) network service, as `kadmin -l`. This resolves the chicken-and-egg problem of trying to connect to the database before it is created. At the `kadmin` prompt, use `init` to create the realm's initial database:

```
# kadmin -l
kadmin> init EXAMPLE.ORG
Realm max ticket life [unlimited]:
```

Lastly, while still in `kadmin`, create the first principal using `add`. Stick to the default options for the principal for now, as these can be changed later with `modify`. Type `?` at the prompt to see the available options.

```
kadmin> add tillman
```

The output should be similar to the following:

```
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
Password: xxxxxxxx
Verifying password - Password: xxxxxxxx
```

Next, start the KDC services by running:

```
# service kdc start
# service kadmind start
```

While there will not be any kerberized daemons running at this point, it is possible to confirm that the KDC is functioning by obtaining a ticket for the principal that was just created:

```
% kinit tillman
```

The output should be similar to the following:

```
tillman@EXAMPLE.ORG's Password:
```

Confirm that a ticket was successfully obtained using `klist`:

```
% klist
```

The output should be similar to the following:

```
Credentials cache: FILE:/tmp/krb5cc_1001
    Principal: tillman@EXAMPLE.ORG

  Issued                Expires               Principal
Aug 27 15:37:58 2013  Aug 28 01:37:58 2013  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
```

The temporary ticket can be destroyed when the test is finished:

```
% kdestroy
```

16.9.2. Configuring a Server to Use Kerberos

The first step in configuring a server to use Kerberos authentication is to ensure that it has the correct configuration in /etc/krb5.conf. The version from the KDC can be used as-is, or it can be regenerated on the new system.

Next, create /etc/krb5.keytab on the server. This is the main part of "Kerberizing" a service - it corresponds to generating a secret shared between the service and the KDC. The secret is a cryptographic key, stored in a "keytab". The keytab contains the server's host key, which allows it and the KDC to verify each others' identity. It must be transmitted to the server in a secure fashion, as the security of the server can be broken if the key is made public. Typically, the keytab is generated on an administrator's trusted machine using `kadmin`, then securely transferred to the server, e.g., with scp(1); it can also be created directly on the server if that is consistent with the desired security policy. It is very important that the keytab is transmitted to the server in a secure fashion: if the key is known by some other party, that party can impersonate any user to the server! Using `kadmin` on the server directly is convenient, because the entry for the host principal in the KDC database is also created using `kadmin`.

Of course, `kadmin` is a kerberized service; a Kerberos ticket is needed to authenticate to the network service, but to ensure that the user running `kadmin` is actually present (and their session has not been hijacked), `kadmin` will prompt for the password to get a fresh ticket. The principal authenticating to the kadmin service must be permitted to use the `kadmin` interface, as specified in /var/heimdal/kadmind.acl. See the section titled "Remote administration" in `info heimdal` for details on designing access control lists. Instead of enabling remote `kadmin` access, the administrator could securely connect to the KDC via the local console or ssh(1), and perform administration locally using `kadmin -l`.

After installing /etc/krb5.conf, use `add --random-key` in `kadmin`. This adds the server's host principal to the database, but does not extract a copy of the host principal key to a keytab. To generate the keytab, use `ext` to extract the server's host principal key to its own keytab:

```
# kadmin
```

The output should be similar to the following:

```
kadmin> add --random-key host/myserver.example.org
Max ticket life [unlimited]:
Max renewable life [unlimited]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []:
kadmin> ext_keytab host/myserver.example.org
kadmin> exit
```

Note that `ext_keytab` stores the extracted key in /etc/krb5.keytab by default. This is good when being run on the server being kerberized, but the `--keytab path/to/file` argument should be used when the keytab is being extracted elsewhere:

```
# kadmin
```

The output should be similar to the following:

```
kadmin> ext_keytab --keytab=/tmp/example.keytab host/myserver.example.org
kadmin> exit
```

The keytab can then be securely copied to the server using scp(1) or a removable media. Be sure to specify a non-default keytab name to avoid inserting unneeded keys into the system's keytab.

At this point, the server can read encrypted messages from the KDC using its shared key, stored in krb5.keytab. It is now ready for the Kerberos-using services to be enabled. One of the most common such services is sshd(8), which supports Kerberos via the GSS-API. In /etc/ssh/sshd_config, add the line:

```
GSSAPIAuthentication yes
```

After making this change, sshd(8) must be restarted for the new configuration to take effect: `service sshd restart`.

16.9.3. Configuring a Client to Use Kerberos

As it was for the server, the client requires configuration in /etc/krb5.conf. Copy the file in place (securely) or re-enter it as needed.

Test the client by using `kinit`, `klist`, and `kdestroy` from the client to obtain, show, and then delete a ticket for an existing principal. Kerberos applications should also be able to connect to Kerberos enabled servers. If that does not work but obtaining a ticket does, the problem is likely with the server and not with the client or the KDC. In the case of kerberized ssh(1), GSS-API is disabled by default, so test using `ssh -o GSSAPIAuthentication=yes hostname`.

When testing a Kerberized application, try using a packet sniffer such as `tcpdump` to confirm that no sensitive information is sent in the clear.

Various Kerberos client applications are available. With the advent of a bridge so that applications using SASL for authentication can use GSS-API mechanisms as well, large classes of client applications can use Kerberos for authentication, from Jabber clients to IMAP clients.

Users within a realm typically have their Kerberos principal mapped to a local user account. Occasionally, one needs to grant access to a local user account to someone who does not have a matching Kerberos principal. For example, `tillman@EXAMPLE.ORG` may need access to the local user account `webdevelopers`. Other principals may also need access to that local account.

The .k5login and .k5users files, placed in a user's home directory, can be used to solve this problem. For example, if the following .k5login is placed in the home directory of `webdevelopers`, both principals listed will have access to that account without requiring a shared password:

```
tillman@example.org
jdoe@example.org
```

Refer to ksu(1) for more information about .k5users.

16.9.4. MIT Differences

The major difference between the MIT and Heimdal implementations is that `kadmin` has a different, but equivalent, set of commands and uses a different protocol. If the KDC is MIT, the Heimdal version of `kadmin` cannot be used to administer the KDC remotely, and vice versa.

Client applications may also use slightly different command line options to accomplish the same tasks. Following the instructions at http://web.mit.edu/Kerberos/www/ is recommended. Be careful of path issues: the MIT port installs into /usr/local/ by default, and the FreeBSD system applications run instead of the MIT versions if `PATH` lists the system directories first.

When using MIT Kerberos as a KDC on FreeBSD, execute the following commands to add the required configurations to /etc/rc.conf:

```
# sysrc kdc_program="/usr/local/sbin/krb5kdc"
# sysrc kadmind_program="/usr/local/sbin/kadmind"
# sysrc kdc_flags=""
# sysrc kdc_enable="YES"
# sysrc kadmind_enable="YES"
```

16.9.5. Kerberos Tips, Tricks, and Troubleshooting

When configuring and troubleshooting Kerberos, keep the following points in mind:

- When using either Heimdal or MITKerberos from ports, ensure that the `PATH` lists the port's versions of the client applications before the system versions.
- If all the computers in the realm do not have synchronized time settings, authentication may fail. "Clock Synchronization with NTP" describes how to synchronize clocks using NTP.
- If the hostname is changed, the `host/` principal must be changed and the keytab updated. This also applies to special keytab entries like the `HTTP/` principal used for Apache's www/mod_auth_kerb.
- All hosts in the realm must be both forward and reverse resolvable in DNS or, at a minimum, exist in /etc/hosts. CNAMEs will work, but the A and PTR records must be correct and in place. The error message for unresolvable hosts is not intuitive: `Kerberos5 refuses authentication because Read req failed: Key table entry not found`.
- Some operating systems that act as clients to the KDC do not set the permissions for `ksu` to be setuid `root`. This means that `ksu` does not work. This is a permissions problem, not a KDC error.
- With MITKerberos, to allow a principal to have a ticket life longer than the default lifetime of ten hours, use `modify_principal` at the kadmin(8) prompt to change the `maxlife` of both the principal in question and the `krbtgt` principal. The principal can then use `kinit -l` to request a ticket with a longer lifetime.
- When running a packet sniffer on the KDC to aid in troubleshooting while running `kinit` from a workstation, the Ticket Granting Ticket (TGT) is sent immediately, even before the password is

typed. This is because the Kerberos server freely transmits a TGT to any unauthorized request. However, every TGT is encrypted in a key derived from the user's password. When a user types their password, it is not sent to the KDC, it is instead used to decrypt the TGT that `kinit` already obtained. If the decryption process results in a valid ticket with a valid time stamp, the user has valid Kerberos credentials. These credentials include a session key for establishing secure communications with the Kerberos server in the future, as well as the actual TGT, which is encrypted with the Kerberos server's own key. This second layer of encryption allows the Kerberos server to verify the authenticity of each TGT.

- Host principals can have a longer ticket lifetime. If the user principal has a lifetime of a week but the host being connected to has a lifetime of nine hours, the user cache will have an expired host principal and the ticket cache will not work as expected.
- When setting up krb5.dict to prevent specific bad passwords from being used as described in kadmind(8), remember that it only applies to principals that have a password policy assigned to them. The format used in krb5.dict is one string per line. Creating a symbolic link to /usr/share/dict/words might be useful.

16.9.6. Mitigating Kerberos Limitations

Since Kerberos is an all or nothing approach, every service enabled on the network must either be modified to work with Kerberos or be otherwise secured against network attacks. This is to prevent user credentials from being stolen and re-used. An example is when Kerberos is enabled on all remote shells but the non-Kerberized POP3 mail server sends passwords in plain text.

The KDC is a single point of failure. By design, the KDC must be as secure as its master password database. The KDC should have absolutely no other services running on it and should be physically secure. The danger is high because Kerberos stores all passwords encrypted with the same master key which is stored as a file on the KDC.

A compromised master key is not quite as bad as one might fear. The master key is only used to encrypt the Kerberos database and as a seed for the random number generator. As long as access to the KDC is secure, an attacker cannot do much with the master key.

If the KDC is unavailable, network services are unusable as authentication cannot be performed. This can be alleviated with a single master KDC and one or more slaves, and with careful implementation of secondary or fall-back authentication using PAM.

Kerberos allows users, hosts and services to authenticate between themselves. It does not have a mechanism to authenticate the KDC to the users, hosts, or services. This means that a trojaned `kinit` could record all user names and passwords. File system integrity checking tools like security/tripwire can alleviate this.

16.9.7. Resources and Further Information

- The Kerberos FAQ
- Designing an Authentication System: a Dialog in Four Scenes
- RFC 4120, The Kerberos Network Authentication Service (V5)
- MIT Kerberos home page

- Heimdal Kerberos project wiki page

16.10. TCP Wrappers

TCP Wrappers is a host-based network access control system. By intercepting incoming network requests before they reach the actual network service, TCP Wrappers assess whether the source IP address is permitted or denied access based on predefined rules in configuration files.

However, while TCP Wrappers provide basic access control, they should not be considered a substitute for more robust security measures. For comprehensive protection, it's recommended to use advanced technologies like firewalls, along with proper user authentication practices and intrusion detection systems.

16.10.1. Initial Configuration

TCP Wrappers are enabled by default in inetd(8). So the first step will be to enable inetd(8) executing the following commands:

```
# sysrc inetd_enable="YES"
# service inetd start
```

Then, properly configure /etc/hosts.allow.

Unlike other implementations of TCP Wrappers, the use of hosts.deny is deprecated in FreeBSD. All configuration options should be placed in /etc/hosts.allow.

In the simplest configuration, daemon connection policies are set to either permit or block, depending on the options in /etc/hosts.allow. The default configuration in FreeBSD is to allow all connections to the daemons started with inetd.

Basic configuration usually takes the form of `daemon : address : action`, where `daemon` is the daemon which inetd started, `address` is a valid hostname, IP address, or an IPv6 address enclosed in brackets ([ ]), and `action` is either `allow` or `deny`. TCP Wrappers uses a first rule match semantic, meaning that the configuration file is scanned from the beginning for a matching rule. When a match is found, the rule is applied and the search process stops.

For example, to allow POP3 connections via the mail/qpopper daemon, the following lines should be appended to /etc/hosts.allow:

```
# This line is required for POP3 connections:
qpopper : ALL : allow
```

Whenever this file is edited, restart inetd:

```
# service inetd restart
```

16.10.2. Advanced Configuration

TCP Wrappers provides advanced options to allow more control over the way connections are handled. In some cases, it may be appropriate to return a comment to certain hosts or daemon connections. In other cases, a log entry should be recorded or an email sent to the administrator. Other situations may require the use of a service for local connections only. This is all possible through the use of configuration options known as wildcards, expansion characters, and external command execution. To learn more about wildcards and their associated functionality, refer to hosts_access(5).

16.11. Access Control Lists

Access Control Lists (ACLs) extend traditional UNIX® file permissions by allowing fine-grained access control for users and groups on a per-file or per-directory basis. Each ACL entry defines a user or group and the associated permissions, such as read, write, and execute. FreeBSD provides commands like getfacl(1) and setfacl(1) to manage ACLs.

ACLs are useful in scenarios requiring more specific access control than standard permissions, commonly used in multi-user environments or shared hosting. However, complexity may be unavoidable, but careful planning is required to ensure that the desired security properties are being provided

FreeBSD supports the implementation of NFSv4 ACLs in both UFS and OpenZFS. Please note that some arguments to the setfacl(1) command only work with POSIX ACLs and others in NFSv4 ACLs.

16.11.1. Enabling ACL Support in UFS

ACLs are enabled by the mount-time administrative flag, `acls`, which may be added to /etc/fstab.

Therefore it will be necessary to access /etc/fstab and in the options section add the `acls` flag as follows:

```
# Device        Mountpoint        FStype    Options     Dump      Pass#
/dev/ada0s1a    /                 ufs       rw,acls     1         1
```

16.11.2. Get ACLs information

It is possible to check the ACLs of a file or a directory using getfacl(1).

For example, to view the ACL settings on ~/test file execute the following command:

```
% getfacl test
```

The output should be similar to the following in case of using NFSv4 ACLs:

```
# file: test
# owner: freebsduser
# group: freebsduser
            owner@:rw-p--aARWcCos:-------:allow
            group@:r-----a-R-c--s:-------:allow
         everyone@:r-----a-R-c--s:-------:allow
```

And the output should be similar to the following in case of using POSIX.1e ACLs:

```
# file: test
# owner: freebsduser
# group: freebsduser
user::rw-
group::r--
other::r--
```

16.11.3. Working with ACLs

setfacl(1) can be used to add, modify or remove ACLs from a file or directory.

As noted above, some arguments to setfacl(1) do not work with NFSv4 ACLs, and vice versa. This section covers how to execute the commands for POSIX ACLs and for NFSv4 ACLs and shows examples of both.

For example, to set the mandatory elements of the POSIX.1e default ACL:

```
% setfacl -d -m u::rwx,g::rx,o::rx,mask::rwx directory
```

This other example sets read, write, and execute permissions for the file owner's POSIX.1e ACL entry and read and write permissions for group mail on file:

```
% setfacl -m u::rwx,g:mail:rw file
```

To do the same as in the previous example but in NFSv4 ACL:

```
% setfacl -m owner@:rwxp::allow,g:mail:rwp::allow file
```

To remove all ACL entries except for the three required from file in POSIX.1e ACL:

```
% setfacl -bn file
```

To remove all ACL entries in NFSv4 ACL:

```
% setfacl -b file
```

Refer to getfacl(1) and setfacl(1) for more information about the options available for these commands.

16.12. Capsicum

Capsicum is a lightweight OS capability and sandbox framework implementing a hybrid capability system model. Capabilities are unforgeable tokens of authority that can be delegated and must be presented to perform an action. Capsicum makes file descriptors into capabilities.

Capsicum can be used for application and library compartmentalisation, the decomposition of larger bodies of software into isolated (sandboxed) components in order to implement security policies and limit the impact of software vulnerabilities.

16.13. Process Accounting

Process accounting is a security method in which an administrator may keep track of system resources used and their allocation among users, provide for system monitoring, and minimally track a user’s commands.

Process accounting has both positive and negative points. One of the positives is that an intrusion may be narrowed down to the point of entry. A negative is the amount of logs generated by process accounting, and the disk space they may require. This section walks an administrator through the basics of process accounting.

If more fine-grained accounting is needed, refer to Security Event Auditing.

16.13.1. Enabling and Utilizing Process Accounting

Before using process accounting, it must be enabled using the following commands:

```
# sysrc accounting_enable=yes
# service accounting start
```

The accounting information is stored in files located in /var/account, which is automatically created, if necessary, the first time the accounting service starts. These files contain sensitive information, including all the commands issued by all users. Write access to the files is limited to `root`, and read access is limited to `root` and members of the `wheel` group. To also prevent members of `wheel` from reading the files, change the mode of the /var/account directory to allow access only by `root`.

Once enabled, accounting will begin to track information such as CPU statistics and executed commands. All accounting logs are in a non-human readable format which can be viewed using sa(8). If issued without any options, sa(8) prints information relating to the number of per-user calls, the total elapsed time in minutes, total CPU and user time in minutes, and the average number of I/O operations. Refer to sa(8) for the list of available options which control the output.

To display the commands issued by users, use `lastcomm`.

For example, this command prints out all usage of `ls` by `trhodes` on the `ttyp1` terminal:

```
# lastcomm ls trhodes ttyp1
```

Many other useful options exist and are explained in lastcomm(1), acct(5), and sa(8).

16.14. Resource Limits

In FreeBSD, resource limits refer to the mechanisms that control and manage the allocation of various system resources to processes and users. These limits are designed to prevent a single process or user from consuming an excessive amount of resources, which could lead to performance degradation or system instability. Resource limits help ensure fair resource distribution among all active processes and users on the system.

FreeBSD provides several methods for an administrator to limit the amount of system resources an individual may use.

The traditional method defines login classes by editing /etc/login.conf. While this method is still supported, any changes require a multi-step process of editing this file, rebuilding the resource database, making necessary changes to /etc/master.passwd, and rebuilding the password database. This can become time consuming, depending upon the number of users to configure.

rctl(8) can be used to provide a more fine-grained method for controlling resource limits. This command supports more than user limits as it can also be used to set resource constraints on processes and jails.

This section demonstrates both methods for controlling resources, beginning with the traditional method.

16.14.1. Types of Resources

FreeBSD provides limits for various types of resources, including:

Table 30. Resource types

Type	Description
CPU Time	Limits the amount of CPU time a process can consume
Memory	Controls the amount of physical memory a process can use

Type	Description
Open Files	Limits the number of files a process can have open simultaneously
Processes	Controls the number of processes a user or a process can create
File Size	Limits the maximum size of files that a process can create
Core Dumps	Controls whether processes are allowed to generate core dump files
Network Resources	Limits the amount of network resources (e.g., sockets) a process can use

For a full listing of types see login.conf(5) and rctl(8).

16.14.2. Configuring Login Classes

In the traditional method, login classes and the resource limits to apply to a login class are defined in /etc/login.conf. Each user account can be assigned to a login class, where `default` is the default login class. Each login class has a set of login capabilities associated with it. A login capability is a `name=value` pair, where *name* is a well-known identifier and *value* is an arbitrary string which is processed accordingly depending on the *name*.

The first step to configure a resource limit will be to open /etc/login.conf by executing the following command:

```
# ee /etc/login.conf
```

Then locate the section for the user class to be modified. In this example, let's assume the user class is named `limited`, create it in case it does not exist.

```
limited:\ ①
        :maxproc=50:\ ②
        :tc=default: ③
```

① Name of the user class.

② Sets the maximum number of processes (maxproc) to 50 for users in the `limited` class.

③ Indicates that this user class inherits the default settings from the "default" class.

After modifying the /etc/login.conf file, run cap_mkdb(1) to generate the database that FreeBSD uses to apply these settings:

```
# cap_mkdb /etc/login.conf
```

chpass(1) can be used to change the class to the desired user by executing the following command:

```
# chpass username
```

This will open a text editor, add the new `limited` class there as follows:

```
#Changing user information for username.
Login: username
Password: $6$2H.419USdGaiJeqK$6kgcTnDadasdasd3YnlNZsOni5AMymibkAfRCPirc7ZFjjv
DVsKyXx26daabdfqSdasdsmL/ZMUpdHiO0
Uid [#]: 1001
Gid [# or name]: 1001
Change [month day year]:
Expire [month day year]:
Class: limited
Home directory: /home/username
Shell: /bin/sh
Full Name: User &
Office Location:
Office Phone:
Home Phone:
Other information:
```

Now, the user assigned to the `limited` class will have a maximum process limit of 50. Remember that this is just one example of setting a resource limit using the /etc/login.conf file.

Keep in mind that after making changes to the /etc/login.conf file, the user needs to log out and log back in for the changes to take effect. Additionally, always exercise caution when editing system configuration files, especially when using privileged access.

16.14.3. Enabling and Configuring Resource Limits

The rctl(8) system provides a more fine-grained way to set and manage resource limits for individual processes and users. It allows dynamically assigning resource limits to specific processes or users, regardless of their user class.

The first step to use rctl(8) will be to enable it adding the following line to /boot/loader.conf and reboot the system:

```
kern.racct.enable=1
```

Then enable and start the rctl(8) service by executing the following commands:

```
# sysrc rctl_enable="YES"
# service rctl start
```

Then rctl(8) may be used to set rules for the system.

Rule syntax (rctl.conf(5)) is controlled through the use of a subject, subject-id, resource, and action, as seen in this example rule:

```
subject:subject-id:resource:action=amount/per
```

For example to constrained the user to add no more than 10 processes execute the following command:

```
# rctl -a user:username:maxproc:deny=10/user
```

To check the applied resource limits the rctl(8) command can be executed:

```
# rctl
```

The output should be similar to the following:

```
user:username:maxproc:deny=10
```

Rules will persist across reboots if they have been added to /etc/rctl.conf. The format is a rule, without the preceding command. For example, the previous rule could be added as:

```
user:username:maxproc:deny=10
```

16.15. Monitoring Third Party Security Issues

In recent years, the security world has made many improvements to how vulnerability assessment is handled. The threat of system intrusion increases as third party utilities are installed and configured for virtually any operating system available today.

Vulnerability assessment is a key factor in security. While FreeBSD releases advisories for the base system, doing so for every third party utility is beyond the FreeBSD Project's capability. There is a way to mitigate third party vulnerabilities and warn administrators of known security issues. A FreeBSD add on utility known as pkg includes options explicitly for this purpose.

pkg polls a database for security issues. The database is updated and maintained by the FreeBSD Security Team and ports developers.

Installation provides periodic(8) configuration files for maintaining the pkg audit database, and provides a programmatic method of keeping it updated.

After installation, and to audit third party utilities as part of the Ports Collection at any time, an administrator may choose to update the database and view known vulnerabilities of installed packages by invoking:

```
% pkg audit -F
```

The output should be similar to the following:

```
vulnxml file up-to-date
chromium-116.0.5845.96_1 is vulnerable:
  chromium -- multiple vulnerabilities
  CVE: CVE-2023-4431
  CVE: CVE-2023-4427
  CVE: CVE-2023-4428
  CVE: CVE-2023-4429
  CVE: CVE-2023-4430
  WWW: https://vuxml.FreeBSD.org/freebsd/5fa332b9-4269-11ee-8290-a8a1599412c6.html

samba413-4.13.17_5 is vulnerable:
  samba -- multiple vulnerabilities
  CVE: CVE-2023-3347
  CVE: CVE-2023-34966
  CVE: CVE-2023-34968
  CVE: CVE-2022-2127
  CVE: CVE-2023-34967
  WWW: https://vuxml.FreeBSD.org/freebsd/441e1e1a-27a5-11ee-a156-080027f5fec9.html

2 problem(s) in 2 installed package(s) found.
```

By pointing a web browser to the displayed URL, an administrator may obtain more information about the vulnerability.

This will include the versions affected, by FreeBSD port version, along with other web sites which may contain security advisories.

16.16. FreeBSD Security Advisories

Like many producers of quality operating systems, the FreeBSD Project has a security team which is responsible for determining the End-of-Life (EoL) date for each FreeBSD release and to provide security updates for supported releases which have not yet reached their EoL. More information about the FreeBSD security team and the supported releases is available on the FreeBSD security page.

One task of the security team is to respond to reported security vulnerabilities in the FreeBSD operating system. Once a vulnerability is confirmed, the security team verifies the steps necessary to fix the vulnerability and updates the source code with the fix. It then publishes the details as a "Security Advisory". Security advisories are published on the FreeBSD website and mailed to the FreeBSD security notifications mailing list, FreeBSD security mailing list, and FreeBSD announcements mailing list.

16.16.1. Format of a Security Advisory

Here is an example of a FreeBSD security advisory:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

=============================================================================
FreeBSD-SA-23:07.bhyve                                      Security Advisory
                                                          The FreeBSD Project

Topic:          bhyve privileged guest escape via fwctl

Category:       core
Module:         bhyve
Announced:      2023-08-01
Credits:        Omri Ben Bassat and Vladimir Eli Tokarev from Microsoft
Affects:        FreeBSD 13.1 and 13.2
Corrected:      2023-08-01 19:48:53 UTC (stable/13, 13.2-STABLE)
                2023-08-01 19:50:47 UTC (releng/13.2, 13.2-RELEASE-p2)
                2023-08-01 19:48:26 UTC (releng/13.1, 13.1-RELEASE-p9)
CVE Name:       CVE-2023-3494

For general information regarding FreeBSD Security Advisories,
including descriptions of the fields above, security branches, and the
following sections, please visit <URL:https://security.FreeBSD.org/>.

I.   Background

bhyve(8)'s fwctl interface provides a mechanism through which guest
firmware can query the hypervisor for information about the virtual
machine.  The fwctl interface is available to guests when bhyve is run
with the "-l bootrom" option, used for example when booting guests in
UEFI mode.

bhyve is currently only supported on the amd64 platform.

II.  Problem Description

The fwctl driver implements a state machine which is executed when the
guest accesses certain x86 I/O ports.  The interface lets the guest copy
a string into a buffer resident in the bhyve process' memory.  A bug in
the state machine implementation can result in a buffer overflowing when
copying this string.

III. Impact

A malicious, privileged software running in a guest VM can exploit the
buffer overflow to achieve code execution on the host in the bhyve
userspace process, which typically runs as root.  Note that bhyve runs
```

```
in a Capsicum sandbox, so malicious code is constrained by the
capabilities available to the bhyve process.

IV.  Workaround

No workaround is available.  bhyve guests that are executed without the
"-l bootrom" option are unaffected.

V.   Solution

Upgrade your vulnerable system to a supported FreeBSD stable or
release / security branch (releng) dated after the correction date.

Perform one of the following:

1) To update your vulnerable system via a binary patch:

Systems running a RELEASE version of FreeBSD on the amd64, i386, or
(on FreeBSD 13 and later) arm64 platforms can be updated via the
freebsd-update(8) utility:

# freebsd-update fetch
# freebsd-update install

Restart all affected virtual machines.

2) To update your vulnerable system via a source code patch:

The following patches have been verified to apply to the applicable
FreeBSD release branches.

a) Download the relevant patch from the location below, and verify the
detached PGP signature using your PGP utility.

[FreeBSD 13.2]
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.2.patch
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.2.patch.asc
# gpg --verify bhyve.13.2.patch.asc

[FreeBSD 13.1]
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.1.patch
# fetch https://security.FreeBSD.org/patches/SA-23:07/bhyve.13.1.patch.asc
# gpg --verify bhyve.13.1.patch.asc

b) Apply the patch.  Execute the following commands as root:

# cd /usr/src
# patch < /path/to/patch

c) Recompile the operating system using buildworld and installworld as
described in <URL:https://www.FreeBSD.org/handbook/makeworld.html>.
```

```
Restart all affected virtual machines.

VI.  Correction details

This issue is corrected by the corresponding Git commit hash or Subversion
revision number in the following stable and release branches:

Branch/path                             Hash                     Revision
- -------------------------------------------------------------------------
stable/13/                              9fe302d78109    stable/13-n255918
releng/13.2/                            2bae613e0da3  releng/13.2-n254625
releng/13.1/                            87702e38a4b4  releng/13.1-n250190
- -------------------------------------------------------------------------

Run the following command to see which files were modified by a
particular commit:

# git show --stat <commit hash>

Or visit the following URL, replacing NNNNNN with the hash:

<URL:https://cgit.freebsd.org/src/commit/?id=NNNNNN>

To determine the commit count in a working tree (for comparison against
nNNNNNN in the table above), run:

# git rev-list --count --first-parent HEAD

VII. References

<URL:https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2023-3494>

The latest revision of this advisory is available at
<URL:https://security.FreeBSD.org/advisories/FreeBSD-SA-23:07.bhyve.asc>
-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEEthUnfoEIffdcgYM7bljekB8AGu8FAmTJdsIACgkQbljekB8A
Gu8Q1Q/7BFw5Aa0cFxBzbdz+O5NAImj58MvKS6xw61bXcYr12jchyT6ENC7yiR+K
qCqbe5TssRbtZ1gg/94gSGEXccz5OcJGxW+qozhcdPUh2L2nzBPkMCrclrYJfTtM
cnmQKjg/wFZLUVr71GEM95ZFaktlZdXyXx9Z8eBzow5rXexpl1TTHQQ2kZZ41K4K
KFhup91dzGCIj02cqbl+1h5BrXJe3s/oNJt5JKIh/GBh5THQu9n6AywQYl18HtjV
fMb1qRTAS9WbiEP5QV2eEuOG86ucuhytqnEN5MnXJ2rLSjfb9izs9HzLo3ggy7yb
hN3tlbfIPjMEwYexieuoyP3rzKkLeYfLXqJU4zKCRnIbBIkMRy4mcFkfcYmI+MhF
NPh2R9kccemppKXeDhKJurH0vsetr8ti+AwOZ3pgO21+9w+mjE+EfaedIi+JWhip
hwqeFv03bAQHJdacNYGV47NsJ91CY4ZgWC3ZOzBZ2Y5SDtKFjyc0bf83WTfU9A/0
drC0z3xaJribah9e6k5d7lmZ7L6aHCbQ70+aayuAEZQLr/N1doB0smNi0IHdrtY0
JdIqmVX+d1ihVhJ05prC460AS/Kolqiaysun1igxR+ZnctE9Xdo1BlLEbYu2KjT4
LpWvSuhRMSQaYkJU72SodQc0FM5mqqNN42Vx+X4EutOfvQuRGlI=
=MlAY
```

```
-----END PGP SIGNATURE-----
```

Every security advisory uses the following format:

- Each security advisory is signed by the PGP key of the Security Officer. The public key for the Security Officer can be verified at OpenPGP Keys.
- The name of the security advisory always begins with `FreeBSD-SA-` (for FreeBSD Security Advisory), followed by the year in two digit format (`23:`), followed by the advisory number for that year (`07.`), followed by the name of the affected application or subsystem (`bhyve`).
- The `Topic` field summarizes the vulnerability.
- The `Category` refers to the affected part of the system which may be one of `core`, `contrib`, or `ports`. The `core` category means that the vulnerability affects a core component of the FreeBSD operating system. The `contrib` category means that the vulnerability affects software included with FreeBSD, such as BIND. The `ports` category indicates that the vulnerability affects software available through the Ports Collection.
- The `Module` field refers to the component location. In this example, the `bhyve` module is affected; therefore, this vulnerability affects an application installed with the operating system.
- The `Announced` field reflects the date the security advisory was published. This means that the security team has verified that the problem exists and that a patch has been committed to the FreeBSD source code repository.
- The `Credits` field gives credit to the individual or organization who noticed the vulnerability and reported it.
- The `Affects` field explains which releases of FreeBSD are affected by this vulnerability.
- The `Corrected` field indicates the date, time, time offset, and releases that were corrected. The section in parentheses shows each branch for which the fix has been merged, and the version number of the corresponding release from that branch. The release identifier itself includes the version number and, if appropriate, the patch level. The patch level is the letter `p` followed by a number, indicating the sequence number of the patch, allowing users to track which patches have already been applied to the system.
- The `CVE Name` field lists the advisory number, if one exists, in the public cve.mitre.org security vulnerabilities database.
- The `Background` field provides a description of the affected module.
- The `Problem Description` field explains the vulnerability. This can include information about the flawed code and how the utility could be maliciously used.
- The `Impact` field describes what type of impact the problem could have on a system.
- The `Workaround` field indicates if a workaround is available to system administrators who cannot immediately patch the system.
- The `Solution` field provides the instructions for patching the affected system. This is a step by step tested and verified method for getting a system patched and working securely.
- The `Correction Details` field displays each affected Subversion or Git branch with the revision number that contains the corrected code.

- The `References` field offers sources of additional information regarding the vulnerability.

Chapter 17. Jails and Containers

17.1. Synopsis

Since system administration is a difficult task, many tools have been developed to make life easier for the administrator. These tools often enhance the way systems are installed, configured, and maintained. One of the tools which can be used to enhance the security of a FreeBSD system is *jails*. Jails have been available since FreeBSD 4.X and continue to be enhanced in their usefulness, performance, reliability, and security.

Jails build upon the chroot(2) concept, which is used to change the root directory of a set of processes. This creates a safe environment, separate from the rest of the system. Processes created in the chrooted environment can not access files or resources outside of it. For that reason, compromising a service running in a chrooted environment should not allow the attacker to compromise the entire system.

However, a chroot has several limitations. It is suited to easy tasks which do not require much flexibility or complex, advanced features. Over time, many ways have been found to escape from a chrooted environment, making it a less than ideal solution for securing services.

Jails improve on the concept of the traditional chroot environment in several ways.

In a traditional chroot environment, processes are only limited in the part of the file system they can access. The rest of the system resources, system users, running processes, and the networking subsystem are shared by the chrooted processes and the processes of the host system. Jails expand this model by virtualizing access to the file system, the set of users, and the networking subsystem. More fine-grained controls are available for tuning the access of a jailed environment. Jails can be considered as a type of operating system-level virtualization.

This chapter covers:

- What a jail is and what purpose it may serve in FreeBSD installations.
- The different types of jail.
- The different ways to configure the network for a jail.
- The jail configuration file.
- How to create the different types of jail.
- How to start, stop, and restart a jail.
- The basics of jail administration, both from inside and outside the jail.
- How to upgrade the different types of jail.
- An incomplete list of the different FreeBSD jail managers.

17.2. Jail Types

Some administrators divide jails into different types, although the underlying technology is the same. Each administrator will have to assess what type of jail to create in each case depending on the problem they have to solve.

Below can be found a list of the different types, their characteristics, and considerations for use.

17.2.1. Thick Jails

A thick jail is a traditional form of FreeBSD Jail. In a thick jail, a complete copy of the base system is replicated within the jail's environment. This means that the jail has its own separate instance of the FreeBSD base system, including libraries, executables, and configuration files. The jail can be thought of as an almost complete standalone FreeBSD installation, but running within the confines of the host system. This isolation ensures that the processes within the jail are kept separate from those on the host and other jails.

Advantages of Thick Jails:

- High degree of isolation: Processes within the jail are isolated from the host system and other jails.
- Independence: Thick jails can have different versions of libraries, configurations, and software than the host system or other jails.
- Security: Since the jail contains its own base system, vulnerabilities or issues affecting the jail environment will not directly impact the host or other jails.

Disadvantages of Thick Jails:

- Resource overhead: Because each jail maintains its own separate base system, thick jails consume more resources compared to thin jails.
- Maintenance: Each jail requires its own maintenance and updates for its base system components.

17.2.2. Thin Jails

A thin jail shares the base system using OpenZFS snapshots or NullFS mounts from a template. Only a minimal subset of base system is duplicated for each thin jail, resulting in less resource consumption compared to a thick jail. However, this also means that thin jails have less isolation and independence compared to thick jails. Changes in shared components could potentially affect multiple thin jails simultaneously.

In summary, a FreeBSD Thin Jail is a type of FreeBSD Jail that replicates a substantial portion, but not all, of the base system within the isolated environment.

Advantages of Thin Jails:

- Resource Efficiency: Thin jails are more resource-efficient compared to thick jails. Since they share most of the base system, they consume less disk space and memory. This makes it possible to run more jails on the same hardware without consuming excessive resources.
- Faster Deployment: Creating and launching thin jails is generally faster compared to thick jails. This can be particularly advantageous when rapidly deploying multiple instances.
- Unified Maintenance: Since thin jails share the majority of their base system with the host

system, updates and maintenance of common base system components (such as libraries and binaries) only need to be done once on the host. This simplifies the maintenance process compared to maintaining an individual base system for each thick jail.

- Shared Resources: Thin jails can more easily share common resources such as libraries and binaries with the host system. This can potentially lead to more efficient disk caching and improved performance for applications within the jail.

Disadvantages of Thin Jails:

- Reduced Isolation: The primary disadvantage of thin jails is that they offer less isolation compared to thick jails. Since they share a significant portion of the template's base system, vulnerabilities or issues affecting shared components could potentially impact multiple jails simultaneously.
- Security Concerns: The reduced isolation in thin jails could pose security risks, as a compromise in one jail might have a greater potential to affect other jails or the host system.
- Dependency Conflicts: If multiple thin jails require different versions of the same libraries or software, managing dependencies can become complex. In some cases, this might require additional effort to ensure compatibility.
- Compatibility Challenges: Applications within a thin jail might encounter compatibility issues if they assume a certain base system environment that differs from the shared components provided by the template.

17.2.3. Service Jails

A service jail shares the complete filesystem tree directly with the host (the jail root path is /) and as such can access and modify any file on the host, and shares the same user accounts with the host. By default it has no access to the network or other resources which are restricted in jails, but they can be configured to re-use the network of the host and to remove some of the jail-restrictions. The use case for service jails is automatic confinement of services/daemons inside a jail with minimal configuration, and without any knowledge of the files needed by such service/daemon. Service jails exist since FreeBSD 15.

Advantages of Service Jails:

- Zero Administration: A service jail ready service needs only one config line in /etc/rc.conf, a service which is not service jails ready needs two config lines.
- Resource Efficiency: Service jails are more resource efficient than thin jails, as they do not need any additional disk space or network resource.
- Faster Deployment: Creating and launching service jails is generally faster compared to thin jails if only distinct services/daemons shall be jailed and no parallel instances of the same service/daemon is needed.
- Shared Resources: Service jails share all resources such as libraries and binaries with the host system. This can potentially lead to more efficient disk caching and improved performance for applications within the jail.
- Process Isolation: Service jails isolate a particular service, it can not see processes which are not a child of the service jail, even if they run within the same user account.

Disadvantages of Service Jails:

- Reduced Isolation: The primary disadvantage of service jails is that they offer no filesystem isolation compared to thick or thin jails.
- Security Concerns: The reduced isolation in service jails could pose security risks, as a compromise in one jail might have a greater potential to affect everything on the host system.

Most of the configuration of jails which is discussed below is not needed for service jails. To understand how jails work, it is recommended to understand those configuration possibilities. The details about what is needed to configure a service jail is in Configuring service jails.

17.2.4. VNET Jails

A FreeBSD VNET jail is a virtualized environment that allows for the isolation and control of network resources for processes running within it. It provides a high level of network segmentation and security by creating a separate network stack for processes within the jail, ensuring that network traffic within the jail is isolated from the host system and other jails.

In essence, FreeBSD VNET jails add a network configuration mechanism. This means a VNET jail can be created as a Thick or Thin Jail.

17.2.5. Linux Jails

A FreeBSD Linux Jail is a feature in the FreeBSD operating system that enables the use of Linux binaries and applications within a FreeBSD jail. This functionality is achieved by incorporating a compatibility layer that allows certain Linux system calls and libraries to be translated and executed on the FreeBSD kernel. The purpose of a Linux Jail is to facilitate the execution of Linux software on a FreeBSD system without needing a separate Linux virtual machine or environment.

17.3. Host Configuration

Before creating any jail on the host system it is necessary to perform certain configuration and obtain some information from the host system.

It will be necessary to configure the jail(8) utility, create the necessary directories to configure and install jails, obtain information from the host's network, and check whether the host uses OpenZFS or UFS as its file system.

> The FreeBSD version running in the jail can not be newer than the version running in the host.

17.3.1. Jail Utility

The jail(8) utility manages jails.

To start jails when the system boots, run the following commands:

```
# sysrc jail_enable="YES"
# sysrc jail_parallel_start="YES"
```

With `jail_parallel_start`, all configured jails will be started in the background.

17.3.2. Networking

Networking for FreeBSD jails can be configured several different ways:

Host Networking Mode (IP Sharing)

In host networking mode, a jail shares the same networking stack as the host system. When a jail is created in host networking mode it uses the same network interface and IP address. This means that the jail does not have a separate IP address, and its network traffic is associated with the host's IP.

Virtual Networks (VNET)

Virtual Networks are a feature of FreeBSD jails that offer more advanced and flexible networking solutions than a basic networking mode like host networking. VNET allows the creation of isolated network stacks for each jail, providing them with their own separate IP addresses, routing tables, and network interfaces. This offers a higher level of network isolation and allows jails to function as if they are running on separate virtual machines.

The netgraph system

netgraph(4) is a versatile kernel framework for creating custom network configurations. It can be used to define how network traffic flows between jails and the host system and between different jails.

17.3.3. Setting Up the Jail Directory Tree

There is no specific place to put the files for the jails.

Some administrators use /jail, others /usr/jail, and still others /usr/local/jails. In this chapter /usr/local/jails will be used.

Apart from /usr/local/jails other directories will be created:

- media will contain the compressed files of the downloaded userlands.
- templates will contain the templates when using Thin Jails.
- containers will contain the jails.

When using OpenZFS, execute the following commands to create datasets for these directories:

```
# zfs create -o mountpoint=/usr/local/jails zroot/jails
# zfs create zroot/jails/media
# zfs create zroot/jails/templates
# zfs create zroot/jails/containers
```

In this case, `zroot` was used for the parent dataset, but other datasets could have been used.

When using UFS, execute the following commands to create the directories:

```
# mkdir /usr/local/jails/
# mkdir /usr/local/jails/media
# mkdir /usr/local/jails/templates
# mkdir /usr/local/jails/containers
```

17.3.4. Jail Configuration Files

There are two ways to configure jails.

The first one is to add an entry for each jail to the file /etc/jail.conf. The other option is to create a file for each jail in the directory /etc/jail.conf.d/.

In case a host system has few jails, an entry for each jail can be added in the file /etc/jail.conf. If the host system has many jails, it is a good idea to have one configuration file for each jail in the /etc/jail.conf.d/ directory.

The files in /etc/jail.conf.d/ must have `.conf` as their extension and have to be included in /etc/jail.conf:

```
.include "/etc/jail.conf.d/*.conf";
```

A typical jail entry would look like this:

```
jailname { ①
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc"; ②
  exec.stop = "/bin/sh /etc/rc.shutdown"; ③
  exec.consolelog = "/var/log/jail_console_${name}.log"; ④

  # PERMISSIONS
  allow.raw_sockets; ⑤
  exec.clean; ⑥
  mount.devfs; ⑦

  # HOSTNAME/PATH
  host.hostname = "${name}"; ⑧
  path = "/usr/local/jails/containers/${name}"; ⑨

  # NETWORK
  ip4.addr = 192.168.1.151; ⑩
  ip6.addr = ::ffff:c0a8:197 ⑪
  interface = em0; ⑫
}
```

① `jailname` - Name of the jail.

② `exec.start` - Command(s) to run in the jail environment when a jail is created. A typical command to run is "/bin/sh /etc/rc".

③ `exec.stop` - Command(s) to run in the jail environment before a jail is removed. A typical command to run is "/bin/sh /etc/rc.shutdown".

④ `exec.consolelog` - A file to direct command output (stdout and stderr) to.

⑤ `allow.raw_sockets` - Allow creating raw sockets inside the jail. Setting this parameter allows utilities like ping(8) and traceroute(8) to operate inside the jail.

⑥ `exec.clean` - Run commands in a clean environment.

⑦ `mount.devfs` - Mount a devfs(5) filesystem on the chrooted /dev directory, and apply the ruleset in the devfs_ruleset parameter to restrict the devices visible inside the jail.

⑧ `host.hostname` - The hostname of the jail.

⑨ `path` - The directory which is to be the root of the jail. Any commands that are run inside the jail, either by jail or from jexec(8), are run from this directory.

⑩ `ip4.addr` - IPv4 address. There are two configuration possibilities for IPv4. The first is to establish an IP or a list of IPs as has been done in the example. The other is to use `ip4` instead and set the `inherit` value to inherit the host's IP address.

⑪ `ip6.addr` - IPv6 address. There are two configuration possibilities for IPv6. The first is to establish an IP or a list of IPs as has been done in the example. The other is to use `ip6` instead and set the `inherit` value to inherit the host's IP address.

⑫ `interface` - A network interface to add the jail's IP addresses. Usually the host interface.

More information about configuration variables can be found in jail(8) and jail.conf(5).

17.4. Classic Jail (Thick Jail)

These jails resemble a real FreeBSD system. They can be managed more or less like a normal host system and updated independently.

17.4.1. Creating a Classic Jail

In principle, a jail only needs a hostname, a root directory, an IP address, and a userland.

The userland for the jail can be obtained from the official FreeBSD download servers.

Execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/15.1-RELEASE/base.txz -o
/usr/local/jails/media/15.1-RELEASE-base.txz
```

Once the download is complete, it will be necessary to extract the contents into the jail directory.

Execute the following commands to extract the userland into the jail's directory:

```
# mkdir -p /usr/local/jails/containers/classic
# tar -xf /usr/local/jails/media/15.1-RELEASE-base.txz -C
/usr/local/jails/containers/classic --unlink
```

With the userland extracted in the jail directory, it will be necessary to copy the timezone and DNS server files:

```
# cp /etc/resolv.conf /usr/local/jails/containers/classic/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/containers/classic/etc/localtime
```

With the files copied, the next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/containers/classic/ fetch install
```

The last step is to configure the jail. It will be necessary to add an entry to the configuration file /etc/jail.conf or in jail.conf.d with the parameters of the jail.

An example would be the following:

```
classic {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/containers/${name}";

  # NETWORK
  ip4.addr = 192.168.1.151;
  interface = em0;
}
```

Execute the following command to start the jail:

```
# service jail start classic
```

More information on how to manage jails can be found in the section Jail Management.

17.5. Thin Jails

Although Thin Jails use the same technology as Thick Jails, the creation procedure is different. Thin jails can be created using OpenZFS snapshots or using templates and NullFS. The use of OpenZFS snapshots and templates using NullFS have certain advantages over classic jails, such as being able to create them faster from snapshots or being able to update multiple jails using NullFS.

17.5.1. Creating a Thin Jail Using OpenZFS Snapshots

Due to the good integration between FreeBSD and OpenZFS it is very easy to create new Thin Jails using OpenZFS Snapshots.

To create a Thin Jail using OpenZFS Snapshots the first step is to create the jail directory tree by following the instructions in Setting up the Jail Directory Tree.

Next, create a template. Templates will only be used to create new jails. For this reason they are created in "read-only" mode so that jails are created with an immutable base.

To create the dataset for the template, execute the following command:

```
# zfs create -p zroot/jails/templates/15.1-RELEASE
```

Then execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/15.1-RELEASE/base.txz -o
/usr/local/jails/media/15.1-RELEASE-base.txz
```

Once the download is complete, it will be necessary to extract the contents in the template directory by executing the following command:

```
# tar -xf /usr/local/jails/media/15.1-RELEASE-base.txz -C
/usr/local/jails/templates/15.1-RELEASE --unlink
```

With the userland extracted in the templates directory, it will be necessary to copy the timezone and DNS server files to the template directory by executing the following command:

```
# cp /etc/resolv.conf /usr/local/jails/templates/15.1-RELEASE/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/templates/15.1-RELEASE/etc/localtime
```

The next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/templates/15.1-RELEASE/ fetch install
```

Once the update is finished, the template is ready.

To create an OpenZFS Snapshot from the template, execute the following command:

```
# zfs snapshot zroot/jails/templates/15.1-RELEASE@base
```

Once the OpenZFS Snapshot has been created, infinite jails can be created using the OpenZFS clone function.

To create a Thin Jail named `thinjail`, execute the following command:

```
# zfs clone zroot/jails/templates/15.1-RELEASE@base zroot/jails/containers/thinjail
```

The last step is to configure the jail. It will be necessary to add an entry to the configuration file /etc/jail.conf or in jail.conf.d with the parameters of the jail.

An example would be the following:

```
thinjail {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/containers/${name}";

  # NETWORK
  ip4 = inherit;
  interface = em0;
}
```

Execute the following command to start the jail:

```
# service jail start thinjail
```

More information on how to manage jails can be found in the section Jail Management.

17.5.2. Creating a Thin Jail Using NullFS

A jail can be created with reduced duplication of system files by using the Thin Jail technique and using NullFS to selectively share specific directories from the host system into the jail.

The first step is to create the dataset to save the template, execute the following command if using OpenZFS:

```
# zfs create -p zroot/jails/templates/15.1-RELEASE-base
```

Or this one if using UFS:

```
# mkdir /usr/local/jails/templates/15.1-RELEASE-base
```

Then execute the following command to download the userland:

```
# fetch https://download.freebsd.org/ftp/releases/amd64/amd64/15.1-RELEASE/base.txz -o
/usr/local/jails/media/15.1-RELEASE-base.txz
```

Once the download is complete, it will be necessary to extract the contents in the template directory by executing the following command:

```
# tar -xf /usr/local/jails/media/15.1-RELEASE-base.txz -C
/usr/local/jails/templates/15.1-RELEASE-base --unlink
```

Once the userland is extracted in the templates directory, it will be necessary to copy the timezone and DNS server files to the template directory by executing the following command:

```
# cp /etc/resolv.conf /usr/local/jails/templates/15.1-RELEASE-base/etc/resolv.conf
# cp /etc/localtime /usr/local/jails/templates/15.1-RELEASE-base/etc/localtime
```

With the files moved to the template, the next thing to do is update to the latest patch level by executing the following command:

```
# freebsd-update -b /usr/local/jails/templates/15.1-RELEASE-base/ fetch install
```

In addition to the base template, it is also necessary to create a directory where the `skeleton` will be located. Some directories will be copied from the template to the `skeleton`.

Execute the following command to create the dataset for the `skeleton` in case of using OpenZFS:

```
# zfs create -p zroot/jails/templates/15.1-RELEASE-skeleton
```

Or this one in case of using UFS:

```
# mkdir /usr/local/jails/templates/15.1-RELEASE-skeleton
```

Then create the `skeleton` directories. The `skeleton` directories will hold the local directories of the jails.

Execute the following commands to create the directories:

```
# mkdir -p /usr/local/jails/templates/15.1-RELEASE-skeleton/home
# mkdir -p /usr/local/jails/templates/15.1-RELEASE-skeleton/usr
# mv /usr/local/jails/templates/15.1-RELEASE-base/etc /usr/local/jails/templates/15.1-
RELEASE-skeleton/etc
# mv /usr/local/jails/templates/15.1-RELEASE-base/usr/local
/usr/local/jails/templates/15.1-RELEASE-skeleton/usr/local
# mv /usr/local/jails/templates/15.1-RELEASE-base/tmp /usr/local/jails/templates/15.1-
RELEASE-skeleton/tmp
# mv /usr/local/jails/templates/15.1-RELEASE-base/var /usr/local/jails/templates/15.1-
RELEASE-skeleton/var
# mv /usr/local/jails/templates/15.1-RELEASE-base/root
/usr/local/jails/templates/15.1-RELEASE-skeleton/root
```

The next step is to create the symlinks to the `skeleton` by executing the following commands:

```
# cd /usr/local/jails/templates/15.1-RELEASE-base/
# mkdir skeleton
# ln -s skeleton/etc etc
# ln -s skeleton/home home
# ln -s skeleton/root root
# ln -s ../skeleton/usr/local usr/local
# ln -s skeleton/tmp tmp
# ln -s skeleton/var var
```

With the `skeleton` ready, it will be necessary to copy the data to the jail directory.

In case of using OpenZFS, OpenZFS snapshots can be used to easily create as many jails as necessary by executing the following commands:

```
# zfs snapshot zroot/jails/templates/15.1-RELEASE-skeleton@base
# zfs clone zroot/jails/templates/15.1-RELEASE-skeleton@base
zroot/jails/containers/thinjail
```

In case of using UFS the cp(1) program can be used by executing the following command:

```
# cp -R /usr/local/jails/templates/15.1-RELEASE-skeleton
```

```
/usr/local/jails/containers/thinjail
```

Then create the directory in which the base template and the skeleton will be mounted:

```
# mkdir -p /usr/local/jails/thinjail-nullfs-base
```

Add a jail entry in /etc/jail.conf or a file in jail.conf.d as follows:

```
thinjail {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/${name}-nullfs-base";

  # NETWORK
  ip4.addr = 192.168.1.153;
  interface = em0;

  # MOUNT
  mount.fstab = "/usr/local/jails/${name}-nullfs-base.fstab";
}
```

Then the create the /usr/local/jails/thinjail-nullfs-base.fstab file as follows:

```
/usr/local/jails/templates/15.1-RELEASE-base  /usr/local/jails/thinjail-nullfs-base/
nullfs   ro          0 0
/usr/local/jails/containers/thinjail     /usr/local/jails/thinjail-nullfs-
base/skeleton nullfs  rw  0 0
```

Execute the following command to start the jail:

```
# service jail start thinjail
```

17.5.3. Creating a VNET Jail

FreeBSD VNET Jails have their own distinct networking stack, including interfaces, IP addresses, routing tables, and firewall rules.

The first step to create a VNET jail is to create the bridge(4) by executing the following command:

```
# ifconfig bridge create
```

The output should be similar to the following:

```
bridge0
```

With the `bridge` created, it will be necessary to attach it to the `em0` interface and bring both of them up by executing the following commands:

```
# ifconfig bridge0 addm em0 up
# ifconfig em0 up
```

To make this setting persist across reboots, add the following lines to /etc/rc.conf:

```
defaultrouter="192.168.1.1"
cloned_interfaces="bridge0"
ifconfig_bridge0="inet 192.168.1.150/24 addm em0 up"
ifconfig_em0="up"
```

For more information on bridging, see Network Bridging.

The next step is to create the jail as indicated above.

Either the Classic Jail (Thick Jail) procedure and the Thin Jails procedure can be used. The only thing that will change is the configuration in the /etc/jail.conf file.

The path /usr/local/jails/containers/vnet will be used as an example for the created jail.

The following is an example configuration for a VNET jail:

```
vnet {
  # STARTUP/LOGGING
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;
  devfs_ruleset = 5;

  # PATH/HOSTNAME
  path = "/usr/local/jails/containers/${name}";
  host.hostname = "${name}";

  # VNET/VIMAGE
  vnet;
  vnet.interface = "${epair}b";

  # NETWORKS/INTERFACES
  $id = "154"; ①
  $ip = "192.168.1.${id}/24";
  $gateway = "192.168.1.1";
  $bridge = "bridge0"; ②
  $epair = "epair${id}";

  # ADD TO bridge INTERFACE
  exec.prestart  = "/sbin/ifconfig ${epair} create up";
  exec.prestart += "/sbin/ifconfig ${epair}a up descr jail:${name}";
  exec.prestart += "/sbin/ifconfig ${bridge} addm ${epair}a up";
  exec.start    += "/sbin/ifconfig ${epair}b ${ip} up";
  exec.start    += "/sbin/route add default ${gateway}";
  exec.start    += "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.poststop = "/sbin/ifconfig ${bridge} deletem ${epair}a";
  exec.poststop += "/sbin/ifconfig ${epair}a destroy";
}
```

① Represents the IP of the Jail, it must be **unique**.

② Refers to the bridge created previously.

17.5.4. Creating a Linux Jail

FreeBSD can run Linux inside a jail using Linux Binary Compatibility and debootstrap(8). Jails do not have a kernel. They run on the host's kernel. Therefore it is necessary to enable Linux Binary Compatibility in the host system.

To enable the Linux ABI at boot time, execute the following command:

```
# sysrc linux_enable="YES"
```

Once enabled, it can be started without rebooting by executing the following command:

```
# service linux start
```

The next step will be to create a jail as indicated above, for example in Creating a Thin Jail Using OpenZFS Snapshots, but **without** performing the configuration. FreeBSD Linux jails require a specific configuration that will be detailed below.

Create an empty /etc/jail.conf to prevent failures when trying to stop the jail:

```
# touch /etc/jail.conf
```

Once the jail has been created as explained above, execute the following command to perform required configuration for the jail and start it:

```
# jail -cm \
    name=ubuntu \
    host.hostname="ubuntu.example.com" \
    path="/usr/local/jails/containers/ubuntu" \
    interface="em0" \
    ip4.addr="192.168.1.150" \
    exec.start="/bin/sh /etc/rc" \
    exec.stop="/bin/sh /etc/rc.shutdown" \
    mount.devfs \
    devfs_ruleset=4 \
    allow.mount \
    allow.mount.devfs \
    allow.mount.fdescfs \
    allow.mount.procfs \
    allow.mount.linprocfs \
    allow.mount.linsysfs \
    allow.mount.tmpfs \
    enforce_statfs=1
```

To access the jail, it will be necessary to install sysutils/debootstrap.

Execute the following command to access the FreeBSD Linux jail:

```
# jexec -u root ubuntu sh
```

Inside the jail, execute the following commands to install pkg(7) and sysutils/debootstrap to prepare the Ubuntu environment:

```
# pkg install debootstrap
# debootstrap jammy /compat/ubuntu
```

When the process has finished and the message `Base system installed successfully` is displayed on the console, it will be necessary to stop the jail from the host system by executing the following command:

```
# service jail onestop ubuntu
```

Then add an entry in /etc/jail.conf for the Linux jail:

```
ubuntu {
  # STARTUP/LOGGING
  exec.start = "/bin/sh /etc/rc";
  exec.stop = "/bin/sh /etc/rc.shutdown";
  exec.consolelog = "/var/log/jail_console_${name}.log";

  # PERMISSIONS
  allow.raw_sockets;
  exec.clean;
  mount.devfs;
  devfs_ruleset = 4;

  # HOSTNAME/PATH
  host.hostname = "${name}";
  path = "/usr/local/jails/containers/${name}";

  # NETWORK
  ip4.addr = 192.168.1.155;
  interface = em0;

  # MOUNT
  mount += "devfs     $path/compat/ubuntu/dev     devfs     rw  0 0";
  mount += "tmpfs     $path/compat/ubuntu/dev/shm tmpfs     rw,size=1g,mode=1777  0
0";
  mount += "fdescfs   $path/compat/ubuntu/dev/fd  fdescfs   rw,linrdlnk 0 0";
  mount += "linprocfs $path/compat/ubuntu/proc    linprocfs rw  0 0";
  mount += "linsysfs  $path/compat/ubuntu/sys     linsysfs  rw  0 0";
  mount += "/tmp      $path/compat/ubuntu/tmp     nullfs    rw  0 0";
  mount += "/home     $path/compat/ubuntu/home    nullfs    rw  0 0";
}
```

Then the jail can be started as usual with the following command:

```
# service jail start ubuntu
```

The Ubuntu environment can be accessed using the following command:

```
# jexec ubuntu chroot /compat/ubuntu /bin/bash
```

More information can be found in the chapter Linux Binary Compatibility.

17.5.5. Configuring Service Jails

A service jail is configured completely via /etc/rc.conf or sysrc(8). The base system services are service jails ready. They contain a config line which enables networking or lift other restrictions of jails. Base system services which do not make sense to run inside jails are configured to not be started as a service jail, even if enabled in /etc/rc.conf. Some examples of such a service are services which want to mount or unmount something in the start of stop method, or only configure something like a route, or firewall, or the like.

Third party services may or may not be service jails ready. To check if a service is service jail ready, the following command can be used:

```
# grep _svcj_options /path/to/rc.d/servicename
```

If there is no output, the service is not service jail ready, or does not need any additional privileges like for example, network access.

If the service is not service jail ready, and needs network access, it can be made ready by adding the necessary config to /etc/rc.conf:

```
# sysrc servicename_svcj_options=net_basic
```

For all possible `_svcj_options` see the rc.conf(5) man-page.

To enable a service jail for a given service, the service needs to be stopped and the `servicename_svcj` variable needs to be set to YES. To put syslogd(8) into a service jail, use the following sequence of commands:

```
# service syslogd stop
# sysrc syslogd_svcj=YES
# service syslogd start
```

If the `servicename_svcj` variable is changed, the service needs to be stopped before it is changed. If it is not stopped, the rc framework will not detect the correct state of the service and will not be able to do what is requested.

Service jails are managed only via rc.conf(5)/sysrc(8) and the service(8) command. The jail utilities, like jls(8) as described in Jail Management can be used to investigate the operation, but the jail(8) command is not supposed to be used to manage them.

17.6. Jail Management

Once the jail is created, there are a number of operations that can be performed, like starting, rebooting or deleting the jail, installing software in it, etc. In this section the different actions that can be done with jails from the host will be described.

17.6.1. List Running Jails

To list the jails that are running on the host system, the command jls(8) can be used:

```
# jls
```

The output should be similar to the following:

```
   JID  IP Address      Hostname                      Path
     1  192.168.250.70  classic
/usr/local/jails/containers/classic
```

jls(8) supports the `--libxo` argument, which through the libxo(3) library allows other types of formats to be displayed, such as `JSON`, `HTML`, etc.

For example, execute the following command to get the `JSON` output:

```
# jls --libxo=json
```

The output should be similar to the following:

```
{"__version": "2", "jail-information": {"jail":
[{"jid":1,"ipv4":"192.168.250.70","hostname":"classic","path":"/usr/local/jails/contai
ners/classic"}]}}
```

17.6.2. Start, Restart, and Stop a Jail

service(8) is used to start, reboot, or stop a jail on the host.

For example, to start a jail, run the following command:

```
# service jail start jailname
```

Change the `start` argument to `restart` or `stop` to perform other actions on the jail.

17.6.3. Destroy a Jail

Destroying a jail is not as simple as stopping the jail using service(8) and removing the jail directory and /etc/jail.conf entry.

FreeBSD takes system security very seriously. For this reason there are certain files that not even the root user can delete. This functionality is known as File Flags.

The first step is to stop the desired jail executing the following command:

```
# service jail stop jailname
```

The second step is to remove these flags with chflags(1) by executing the following command, in which `classic` is the name of the jail to remove:

```
# chflags -R 0 /usr/local/jails/containers/classic
```

The third step is to delete the directory where the jail was:

```
# rm -rf /usr/local/jails/containers/classic
```

Finally, it will be necessary to remove the jail entry in /etc/jail.conf or in jail.conf.d.

17.6.4. Handle Packages in a Jail

The pkg(8) tool supports the `-j` argument in order to handle packages installed inside the jail.

For example, to install www/nginx-lite in the jail, the next command can be executed **from the host**:

```
# pkg -j classic install nginx-lite
```

For more information on working with packages in FreeBSD, see Installing Applications: Packages and Ports.

17.6.5. Access a Jail

While it has been stated above that it is best to manage jails from the host system, a jail can be entered with jexec(8).

The jail can be entered by running jexec(8) from the host:

```
# jexec -u root jailname
```

When gaining access to the jail, the message configured in motd(5) will be displayed.

17.6.6. Execute Commands in a Jail

To execute a command from the host system in a jail the jexec(8) can be used.

For example, to stop a service that is running inside a jail, the command will be executed:

```
# jexec -l jailname service nginx stop
```

17.7. Jail Upgrading

Upgrading FreeBSD Jails ensures that the isolated environments remain secure, up-to-date, and in line with the latest features and improvements available in the FreeBSD ecosystem.

17.7.1. Upgrading a Classic Jail or a Thin Jail using OpenZFS Snapshots

Jails **must be updated from the host** operating system. The default behavior in FreeBSD is to disallow the use of chflags(1) in a jail. This will prevent the update of some files so updating from within the jail will fail.

To update the jail to the latest patch release of the version of FreeBSD it is running, execute the following commands on the host:

```
# freebsd-update -j classic fetch install
# service jail restart classic
```

To upgrade the jail to a new major or minor version, first upgrade the host system as described in Performing Major and Minor Version Upgrades. Once the host has been upgraded and rebooted, the jail can then be upgraded.

In case of upgrade from one version to another, it is easier to create a new jail than to upgrade completely.

For example to upgrade from 13.1-RELEASE to 13.2-RELEASE, execute the following commands on the host:

```
# freebsd-update -j classic -r 13.2-RELEASE upgrade
# freebsd-update -j classic install
# service jail restart classic
# freebsd-update -j classic install
# service jail restart classic
```

It is necessary to execute the `install` step two times. The first one upgrades the kernel, and the second one upgrades the rest of the components.

Then, if it was a major version upgrade, reinstall all installed packages and restart the jail again. This is required because the ABI version changes when upgrading between major versions of FreeBSD.

From the host:

```
# pkg -j jailname upgrade -f
# service jail restart jailname
```

17.7.2. Upgrading a Thin Jail Using NullFS

Since Thin Jails that use NullFS share the majority of system directories, they are very easy to update. It is enough to update the template. This allows updating multiple jails at the same time.

To update the template to the latest patch release of the version of FreeBSD it is running, execute the following commands on the host:

```
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ fetch install
# service jail restart
```

To upgrade the template to a new major or minor version, first upgrade the host system as described in Performing Major and Minor Version Upgrades. Once the host has been upgraded and rebooted, the template can then be upgraded.

For example, to upgrade from 13.1-RELEASE to 13.2-RELEASE, execute the following commands on the host:

```
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ -r 13.2-RELEASE
upgrade
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ install
# service jail restart
# freebsd-update -b /usr/local/jails/templates/13.1-RELEASE-base/ install
# service jail restart
```

17.8. Jail Resource Limits

Controlling the resources that a jail uses from the host system is a task to be taken into account by the system administrator.

Use rctl(8) to manage the resources that a jail can use from the host system.

The `kern.racct.enable` tunable must be enabled at /boot/loader.conf.

The syntax to limit the resources of a jail is as follows:

```
rctl -a jail:<jailname>:resource:action=amount/percentage
```

For example, to limit the maximum RAM that a jail can access, run the following command:

```
# rctl -a jail:classic:memoryuse:deny=2G
```

To make the limitation persistent across reboots of the host system, it will be necessary to add the rule to the /etc/rctl.conf file as follows:

```
jail:classic:memoryuse:deny=2G/jail
```

More information on resource limits can be found in the security chapter in the Resource Limits section.

17.9. Jail Managers and Containers

As previously explained, each type of FreeBSD Jail can be created and configured manually, but FreeBSD also has third-party utilities to make configuration and administration easier.

Below is an incomplete list of the different FreeBSD Jail managers:

Table 31. Jail Managers

Name	License	Package	Documentation
BastilleBSD	BSD-3	sysutils/bastille	Documentation
pot	BSD-3	sysutils/pot	Documentation
cbsd	BSD-2	sysutils/cbsd	Documentation
AppJail	BSD-3	sysutils/appjail, for devel sysutils/appjail-devel	Documentation
iocage	BSD-2	sysutils/iocage	Documentation
ezjail	Beer Ware	sysutils/ezjail	Documentation

Chapter 18. OCI Containers

18.1. Synopsis

The Open Container Initiative, commonly referred to as `OCI`, provides a vendor and OS-agnostic way to describe, distribute, and run containers. The OCI specifications provide these in a way that can be used on many different operating systems, including FreeBSD. The underlying virtualization technology is still FreeBSD jails, with the same feature set, but OCI tooling enables additional ways of working, and constructing, container-based workloads. 14.3-RELEASE and upwards, including snapshots, now include OCI-compatible images, and the Podman toolkit on FreeBSD is ready to use them, on both amd64 and arm64 architectures.

For FreeBSD users familiar with jails, there is a loose analogy:

- FreeBSD's `base.txz` tarball is an example of a container image.
- the `jail.conf` file describes the desired container properties, or `Containerfile`.
- use the `jail ..` command to run a container, given a filesystem path, with the `podman` suite of tools.

By importing this container stack, FreeBSD users both benefit from common tooling, but also enjoy wide support across public and private container registries, and container-specific tooling and services. In the aarch64 and amd64 download directories, you'll see official OCI-format images. The naming may be a little confusing at first, but should make sense once you start using them.

The same images are also available through common public container registries, including Docker Hub, and Github Container Registry, but for the strongest chain of trust, you should fetch your image directly from Official FreeBSD Releases, and import them to your local system.

Each image comprises a subset of a standard FreeBSD `base.txz` release tarball, for various use cases, and the usual FreeBSD `CHECKSUM.*` files, that can be verified against the PGP-signed release announcement. This gives a very strong chain of provenance, directly verifiable from the FreeBSD release team.

```
CHECKSUM.SHA256
CHECKSUM.SHA512
FreeBSD-15.1-RELEASE-amd64-container-image-static.txz
FreeBSD-15.1-RELEASE-amd64-container-image-dynamic.txz
FreeBSD-15.1-RELEASE-amd64-container-image-runtime.txz
FreeBSD-15.1-RELEASE-amd64-container-image-notoolchain.txz
FreeBSD-15.1-RELEASE-amd64-container-image-toolchain.txz
```

18.2. Container Image Naming and Tagging

18.2.1. Introduction

FreeBSD's official container images are built from base system packages, themselves built during the FreeBSD release process, and published to public registries. This guide explains how images are named and tagged, helping you choose the right image for your needs, and ensuring you understand the implications of each choice, as new images are published, and in some cases, mutable tags are amended. This naming convention aligns with the new base system package naming scheme, giving clear provenance between container images and the corresponding FreeBSD releases.

18.2.2. Available Image Types

Five types of FreeBSD container images are available, for `amd64` and `aarch64` architectures.

`freebsd-static`

The static image is intended as a base image, for a workload which is entirely statically linked. It contains no libraries, nor binaries, just the supporting files that most applications of this nature require.

- public TLS certificates
- minimal password file
- minimal termcap
- timezone database

Note that it has, by design, a lean footprint, to make a security compromise of the container less useful to the attacker. There's no UNIX shell, no command-line tools, no dynamic libraries, nor package manager. It is the smallest image provided by the FreeBSD project.

`freebsd-dynamic`

The dynamic image uses the static image as a parent layer, and supports using shared libraries, including `libc`. Most FreeBSD software should run without issue with this image, with minor changes. It doesn't have a shell, rc system, nor a package manager. These limitations are additional security, making it awkward for attackers in a compromised container to move laterally, or make use of tools that were never installed.

`freebsd-runtime`

Again, runtime builds on the preceding dynamic layer, and finally adds the minimum that a user would expect - a UNIX shell, rc system, and the package manager. It is the ideal base image for porting existing applications with a minimum of changes. Users will need to include additional FreeBSD base system libraries, as well as additional packages from the Ports tree. This is the closest to a typical FreeBSD system, including the pkg(8) tool, allowing users to install, or do, almost anything, similar to a non-containerised system.

`freebsd-notoolchain`

This base image contains almost all tools one would expect on a typical FreeBSD system, excluding those that are directly hardware-related, and thus not generally useful within a container, and the compiler and related toolchain, as it is quite large.

`freebsd-toolchain`

The Toolchain base image is the sum of all preceding images, including a full compiler and toolchain. It is generally possible to compile almost any software for FreeBSD in the same way as a normal non-jailed FreeBSD system. All images follow a consistent naming pattern, derived from the FreeBSD release they are based upon.

18.2.3. Image Tag Structure

Immutable Tags

Immutable tags never change and are ideal for production systems where you need stability and predictability. These are the most common tags used by the release process. They will not change, even after FreeBSD security patches, or errata notices. They are ideal for base images for software deployments where a high degree of reproducibility is expected, even as a trade-off against more pro-active security patching.

- `major.minor` (e.g., `14.4`): Points to a specific FreeBSD RELEASE version
- `major.minor.pX` (e.g., `14.4p1`): Points to a specific patch level of a RELEASE
- `major.snapYYYYMMDDHHMMSS` (e.g., `16.snap20260325035941`): a snapshot build with timestamp

Mutable Tags

Mutable tags are updated over time and are ideal for production, or CI/CD systems that always want the latest updates, but still wish to align with FreeBSD ABI compatibility.

- `major` (e.g., `15`) will always be the latest support RELEASE version along the 15.x stable ABI
- `major.snap` (e.g., `15.snap`) follows the latest developer snapshot along the 15.x stable ABI train, will often be in advance of official releases, and carries a risk of breakage, incompatibility, or other unanticipated issues. This is the most mutable tag, and should be used with caution. They are exactly the same as using FreeBSD’s STABLE or CURRENT snapshots, as they use those images as their source.

18.2.4. When to Use Each Tag Type

For Production

Use immutable tags to ensure your environment stays consistent:

- `freebsd-runtime:15.0` - pinned to FreeBSD 15.0-RELEASE

When security patches are released, you must explicitly rebuild or update with a new image tag.

For Continuous Integration and Development

Use mutable tags to automatically get the latest updates:

- `freebsd-runtime:15` - always the latest official RELEASE in the 15.x series

These additional mutable tags will follow the regular published snapshots of FreeBSD STABLE and

CURRENT. Use these if you want to test against what subsequent future releases of FreeBSD may contain, and accept occasional breakage or unanticipated changes.

- `freebsd-runtime:14.snap` - always the latest build from 14-STABLE branch
- `freebsd-runtime:15.snap` - always the latest build from 15-STABLE branch
- `freebsd-runtime:16.snap` - always the latest build from 16-CURRENT branch

Chapter 19. Mandatory Access Control

19.1. Synopsis

FreeBSD supports security extensions based on the POSIX®.1e draft. These security mechanisms include file system Access Control Lists ("Access Control Lists") and Mandatory Access Control (MAC). MAC allows access control modules to be loaded in order to implement security policies. Some modules provide protections for a narrow subset of the system, hardening a particular service. Others provide comprehensive labeled security across all subjects and objects. The mandatory part of the definition indicates that enforcement of controls is performed by administrators and the operating system. This is in contrast to the default security mechanism of Discretionary Access Control (DAC) where enforcement is left to the discretion of users.

This chapter focuses on the MAC framework and the set of pluggable security policy modules FreeBSD provides for enabling various security mechanisms.

Read this chapter to learn:

- The terminology associated with the MAC framework.
- The capabilities of MAC security policy modules as well as the difference between a labeled and non-labeled policy.
- The considerations to take into account before configuring a system to use the MAC framework.
- Which MAC security policy modules are included in FreeBSD and how to configure them.
- How to implement a more secure environment using the MAC framework.
- How to test the MAC configuration to ensure the framework has been properly implemented.

Before reading this chapter:

- Understand UNIX® and FreeBSD basics (FreeBSD Basics).
- Have some familiarity with security and how it pertains to FreeBSD (Security).

> Improper MAC configuration may cause loss of system access, aggravation of users, or inability to access the features provided by Xorg. More importantly, MAC should not be relied upon to completely secure a system. The MAC framework only augments an existing security policy. Without sound security practices and regular security checks, the system will never be completely secure.
>
> The examples contained within this chapter are for demonstration purposes and the example settings should *not* be implemented on a production system. Implementing any security policy takes a good deal of understanding, proper design, and thorough testing.

While this chapter covers a broad range of security issues relating to the MAC framework, the development of new MAC security policy modules will not be covered. A number of security policy modules included with the MAC framework have specific characteristics which are provided for both testing and new module development. Refer to mac_test(4), mac_stub(4) and mac_none(4) for more information on these security policy modules and the various mechanisms they provide.

19.2. Key Terms

The following key terms are used when referring to the MAC framework:

- *compartment*: a set of programs and data to be partitioned or separated, where users are given explicit access to specific component of a system. A compartment represents a grouping, such as a work group, department, project, or topic. Compartments make it possible to implement a need-to-know-basis security policy.
- *integrity*: the level of trust which can be placed on data. As the integrity of the data is elevated, so does the ability to trust that data.
- *level*: the increased or decreased setting of a security attribute. As the level increases, its security is considered to elevate as well.
- *label*: a security attribute which can be applied to files, directories, or other items in the system. It could be considered a confidentiality stamp. When a label is placed on a file, it describes the security properties of that file and will only permit access by files, users, and resources with a similar security setting. The meaning and interpretation of label values depends on the policy configuration. Some policies treat a label as representing the integrity or secrecy of an object while other policies might use labels to hold rules for access.
- *multilabel*: this property is a file system option which can be set in single-user mode using tunefs(8), during boot using fstab(5), or during the creation of a new file system. This option permits an administrator to apply different MAC labels on different objects. This option only applies to security policy modules which support labeling.
- *single label*: a policy where the entire file system uses one label to enforce access control over the flow of data. Whenever `multilabel` is not set, all files will conform to the same label setting.
- *object*: an entity through which information flows under the direction of a *subject*. This includes directories, files, fields, screens, keyboards, memory, magnetic storage, printers or any other data storage or moving device. An object is a data container or a system resource. Access to an object effectively means access to its data.
- *subject*: any active entity that causes information to flow between *objects* such as a user, user process, or system process. On FreeBSD, this is almost always a thread acting in a process on behalf of a user.
- *policy*: a collection of rules which defines how objectives are to be achieved. A policy usually documents how certain items are to be handled. This chapter considers a policy to be a collection of rules which controls the flow of data and information and defines who has access to that data and information.
- *high-watermark*: this type of policy permits the raising of security levels for the purpose of accessing higher level information. In most cases, the original level is restored after the process is complete. Currently, the FreeBSD MAC framework does not include this type of policy.
- *low-watermark*: this type of policy permits lowering security levels for the purpose of accessing information which is less secure. In most cases, the original security level of the user is restored after the process is complete. The only security policy module in FreeBSD to use this is mac_lomac(4).

- *sensitivity*: usually used when discussing Multilevel Security (MLS). A sensitivity level describes how important or secret the data should be. As the sensitivity level increases, so does the importance of the secrecy, or confidentiality, of the data.

19.3. Understanding MAC Labels

A MAC label is a security attribute which may be applied to subjects and objects throughout the system. When setting a label, the administrator must understand its implications in order to prevent unexpected or undesired behavior of the system. The attributes available on an object depend on the loaded policy module, as policy modules interpret their attributes in different ways.

The security label on an object is used as a part of a security access control decision by a policy. With some policies, the label contains all of the information necessary to make a decision. In other policies, the labels may be processed as part of a larger rule set.

There are two types of label policies: single label and multi label. By default, the system will use single label. The administrator should be aware of the pros and cons of each in order to implement policies which meet the requirements of the system's security model.

A single label security policy only permits one label to be used for every subject or object. Since a single label policy enforces one set of access permissions across the entire system, it provides lower administration overhead, but decreases the flexibility of policies which support labeling. However, in many environments, a single label policy may be all that is required.

A single label policy is somewhat similar to DAC as `root` configures the policies so that users are placed in the appropriate categories and access levels. A notable difference is that many policy modules can also restrict `root`. Basic control over objects will then be released to the group, but `root` may revoke or modify the settings at any time.

When appropriate, a multi label policy can be set on a UFS file system by passing `multilabel` to tunefs(8). A multi label policy permits each subject or object to have its own independent MAC label. The decision to use a multi label or single label policy is only required for policies which implement the labeling feature, such as `biba`, `lomac`, and `mls`. Some policies, such as `seeotheruids`, `portacl` and `partition`, do not use labels at all.

Using a multi label policy on a partition and establishing a multi label security model can increase administrative overhead as everything in that file system has a label. This includes directories, files, and even device nodes.

The following command will set `multilabel` on the specified UFS file system. This may only be done in single-user mode and is not a requirement for the swap file system:

```
# tunefs -l enable /
```

Some users have experienced problems with setting the `multilabel` flag on the root partition. If this is the case, please review Troubleshooting the MAC Framework.

Since the multi label policy is set on a per-file system basis, a multi label policy may not be needed if the file system layout is well designed. Consider an example security MAC model for a FreeBSD web server. This machine uses the single label, `biba/high`, for everything in the default file systems. If the web server needs to run at `biba/low` to prevent write up capabilities, it could be installed to a separate UFS /usr/local file system set at `biba/low`.

19.3.1. Label Configuration

Virtually all aspects of label policy module configuration will be performed using the base system utilities. These commands provide a simple interface for object or subject configuration or the manipulation and verification of the configuration.

All configuration may be done using `setfmac`, which is used to set MAC labels on system objects, and `setpmac`, which is used to set the labels on system subjects. For example, to set the `biba` MAC label to `high` on test:

```
# setfmac biba/high test
```

If the configuration is successful, the prompt will be returned without error. A common error is `Permission denied` which usually occurs when the label is being set or modified on a restricted object. Other conditions may produce different failures. For instance, the file may not be owned by the user attempting to relabel the object, the object may not exist, or the object may be read-only. A mandatory policy will not allow the process to relabel the file, maybe because of a property of the file, a property of the process, or a property of the proposed new label value. For example, if a user running at low integrity tries to change the label of a high integrity file, or a user running at low integrity tries to change the label of a low integrity file to a high integrity label, these operations will fail.

The system administrator may use `setpmac` to override the policy module's settings by assigning a different label to the invoked process:

```
# setfmac biba/high test
Permission denied
# setpmac biba/low setfmac biba/high test
# getfmac test
test: biba/high
```

For currently running processes, such as sendmail, `getpmac` is usually used instead. This command takes a process ID (PID) in place of a command name. If users attempt to manipulate a file not in their access, subject to the rules of the loaded policy modules, the `Operation not permitted` error will be displayed.

19.3.2. Predefined Labels

A few FreeBSD policy modules which support the labeling feature offer three predefined labels: `low`, `equal`, and `high`, where:

- `low` is considered the lowest label setting an object or subject may have. Setting this on objects or subjects blocks their access to objects or subjects marked high.

- `equal` sets the subject or object to be disabled or unaffected and should only be placed on objects considered to be exempt from the policy.
- `high` grants an object or subject the highest setting available in the Biba and MLS policy modules.

Such policy modules include mac_biba(4), mac_mls(4) and mac_lomac(4). Each of the predefined labels establishes a different information flow directive. Refer to the manual page of the module to determine the traits of the generic label configurations.

19.3.3. Numeric Labels

The Biba and MLS policy modules support a numeric label which may be set to indicate the precise level of hierarchical control. This numeric level is used to partition or sort information into different groups of classification, only permitting access to that group or a higher group level. For example:

```
biba/10:2+3+6(5:2+3-20:2+3+4+5+6)
```

may be interpreted as "Biba Policy Label/Grade 10:Compartments 2, 3 and 6: (grade 5 ...")

In this example, the first grade would be considered the effective grade with effective compartments, the second grade is the low grade, and the last one is the high grade. In most configurations, such fine-grained settings are not needed as they are considered to be advanced configurations.

System objects only have a current grade and compartment. System subjects reflect the range of available rights in the system, and network interfaces, where they are used for access control.

The grade and compartments in a subject and object pair are used to construct a relationship known as *dominance*, in which a subject dominates an object, the object dominates the subject, neither dominates the other, or both dominate each other. The "both dominate" case occurs when the two labels are equal. Due to the information flow nature of Biba, a user has rights to a set of compartments that might correspond to projects, but objects also have a set of compartments. Users may have to subset their rights using `su` or `setpmac` in order to access objects in a compartment from which they are not restricted.

19.3.4. User Labels

Users are required to have labels so that their files and processes properly interact with the security policy defined on the system. This is configured in /etc/login.conf using login classes. Every policy module that uses labels will implement the user class setting.

To set the user class default label which will be enforced by MAC, add a `label` entry. An example `label` entry containing every policy module is displayed below. Note that in a real configuration, the administrator would never enable every policy module. It is recommended that the rest of this chapter be reviewed before any configuration is implemented.

```
default:\
    :welcome=/etc/motd:\
    :setenv=MAIL=/var/mail/$,BLOCKSIZE=K:\
    :path=~/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin:\
    :manpath=/usr/share/man /usr/local/man:\
    :nologin=/usr/sbin/nologin:\
    :cputime=1h30m:\
    :datasize=8M:\
    :vmemoryuse=100M:\
    :stacksize=2M:\
    :memorylocked=4M:\
    :memoryuse=8M:\
    :filesize=8M:\
    :coredumpsize=8M:\
    :openfiles=24:\
    :maxproc=32:\
    :priority=0:\
    :requirehome:\
    :passwordtime=91d:\
    :umask=022:\
    :ignoretime@:\
    :label=partition/13,mls/5,biba/10(5-15),lomac/10[2]:
```

While users can not modify the default value, they may change their label after they login, subject to the constraints of the policy. The example above tells the Biba policy that a process's minimum integrity is `5`, its maximum is `15`, and the default effective label is `10`. The process will run at `10` until it chooses to change label, perhaps due to the user using `setpmac`, which will be constrained by Biba to the configured range.

After any change to login.conf, the login class capability database must be rebuilt using `cap_mkdb`.

Many sites have a large number of users requiring several different user classes. In depth planning is required as this can become difficult to manage.

19.3.5. Network Interface Labels

Labels may be set on network interfaces to help control the flow of data across the network. Policies using network interface labels function in the same way that policies function with respect to objects. Users at high settings in Biba, for example, will not be permitted to access network interfaces with a label of `low`.

When setting the MAC label on network interfaces, `maclabel` may be passed to `ifconfig`:

```
# ifconfig bge0 maclabel biba/equal
```

This example will set the MAC label of `biba/equal` on the `bge0` interface. When using a setting similar to `biba/high(low-high)`, the entire label should be quoted to prevent an error from being returned.

Each policy module which supports labeling has a tunable which may be used to disable the MAC label on network interfaces. Setting the label to `equal` will have a similar effect. Review the output of `sysctl`, the policy manual pages, and the information in the rest of this chapter for more information on those tunables.

19.4. Planning the Security Configuration

Before implementing any MAC policies, a planning phase is recommended. During the planning stages, an administrator should consider the implementation requirements and goals, such as:

- How to classify information and resources available on the target systems.
- Which information or resources to restrict access to along with the type of restrictions that should be applied.
- Which MAC modules will be required to achieve this goal.

A trial run of the trusted system and its configuration should occur *before* a MAC implementation is used on production systems. Since different environments have different needs and requirements, establishing a complete security profile will decrease the need of changes once the system goes live.

Consider how the MAC framework augments the security of the system as a whole. The various security policy modules provided by the MAC framework could be used to protect the network and file systems or to block users from accessing certain ports and sockets. Perhaps the best use of the policy modules is to load several security policy modules at a time in order to provide a MLS environment. This approach differs from a hardening policy, which typically hardens elements of a system which are used only for specific purposes. The downside to MLS is increased administrative overhead.

The overhead is minimal when compared to the lasting effect of a framework which provides the ability to pick and choose which policies are required for a specific configuration and which keeps performance overhead down. The reduction of support for unneeded policies can increase the overall performance of the system as well as offer flexibility of choice. A good implementation would consider the overall security requirements and effectively implement the various security policy modules offered by the framework.

A system utilizing MAC guarantees that a user will not be permitted to change security attributes at will. All user utilities, programs, and scripts must work within the constraints of the access rules provided by the selected security policy modules and control of the MAC access rules is in the hands of the system administrator.

It is the duty of the system administrator to carefully select the correct security policy modules. For an environment that needs to limit access control over the network, the mac_portacl(4), mac_ifoff(4), and mac_biba(4) policy modules make good starting points. For an environment where strict confidentiality of file system objects is required, consider the mac_bsdextended(4) and mac_mls(4) policy modules.

Policy decisions could be made based on network configuration. If only certain users should be permitted access to ssh(1), the mac_portacl(4) policy module is a good choice. In the case of file systems, access to objects might be considered confidential to some users, but not to others. As an example, a large development team might be broken off into smaller projects where developers in project A might not be permitted to access objects written by developers in project B. Yet both projects might need to access objects created by developers in project C. Using the different security policy modules provided by the MAC framework, users could be divided into these groups and then given access to the appropriate objects.

Each security policy module has a unique way of dealing with the overall security of a system. Module selection should be based on a well thought out security policy which may require revision and reimplementation. Understanding the different security policy modules offered by the MAC framework will help administrators choose the best policies for their situations.

The rest of this chapter covers the available modules, describes their use and configuration, and in some cases, provides insight on applicable situations.

> Implementing MAC is much like implementing a firewall since care must be taken to prevent being completely locked out of the system. The ability to revert back to a previous configuration should be considered and the implementation of MAC over a remote connection should be done with extreme caution.

19.5. Available MAC Policies

The default FreeBSD kernel includes `options MAC`. This means that every module included with the MAC framework can be loaded with `kldload` as a run-time kernel module. After testing the module, add the module name to /boot/loader.conf so that it will load during boot. Each module also provides a kernel option for those administrators who choose to compile their own custom kernel.

FreeBSD includes a group of policies that will cover most security requirements. Each policy is summarized below. The last three policies support integer settings in place of the three default labels.

19.5.1. The MAC See Other UIDs Policy

Module name: mac_seeotheruids.ko

Kernel configuration line: `options MAC_SEEOTHERUIDS`

Boot option: `mac_seeotheruids_load="YES"`

The mac_seeotheruids(4) module extends the `security.bsd.see_other_uids` and `security.bsd.see_other_gids sysctl` tunables. This option does not require any labels to be set before configuration and can operate transparently with other modules.

After loading the module, the following `sysctl` tunables may be used to control its features:

- `security.mac.seeotheruids.enabled` enables the module and implements the default settings which deny users the ability to view processes and sockets owned by other users.
- `security.mac.seeotheruids.specificgid_enabled` allows specified groups to be exempt from this policy. To exempt specific groups, use the `security.mac.seeotheruids.specificgid=XXX sysctl` tunable, replacing *XXX* with the numeric group ID to be exempted.

- `security.mac.seeotheruids.primarygroup_enabled` is used to exempt specific primary groups from this policy. When using this tunable, `security.mac.seeotheruids.specificgid_enabled` may not be set.

19.5.2. The MAC BSD Extended Policy

Module name: mac_bsdextended.ko

Kernel configuration line: `options MAC_BSDEXTENDED`

Boot option: `mac_bsdextended_load="YES"`

The mac_bsdextended(4) module enforces a file system firewall. It provides an extension to the standard file system permissions model, permitting an administrator to create a firewall-like ruleset to protect files, utilities, and directories in the file system hierarchy. When access to a file system object is attempted, the list of rules is iterated until either a matching rule is located or the end is reached. This behavior may be changed using `security.mac.bsdextended.firstmatch_enabled`. Similar to other firewall modules in FreeBSD, a file containing the access control rules can be created and read by the system at boot time using an rc.conf(5) variable.

The rule list may be entered using ugidfw(8) which has a syntax similar to ipfw(8). More tools can be written by using the functions in the libugidfw(3) library.

After the mac_bsdextended(4) module has been loaded, the following command may be used to list the current rule configuration:

```
# ugidfw list
0 slots, 0 rules
```

By default, no rules are defined and everything is completely accessible. To create a rule which blocks all access by users but leaves `root` unaffected:

```
# ugidfw add subject not uid root new object not uid root mode n
```

While this rule is simple to implement, it is a very bad idea as it blocks all users from issuing any commands. A more realistic example blocks `user1` all access, including directory listings, to `user2`'s home directory:

```
# ugidfw set 2 subject uid user1 object uid user2 mode n
# ugidfw set 3 subject uid user1 object gid user2 mode n
```

Instead of `user1`, `not uid user2` could be used in order to enforce the same access restrictions for all users. However, the `root` user is unaffected by these rules.

> Extreme caution should be taken when working with this module as incorrect use could block access to certain parts of the file system.

19.5.3. The MAC Interface Silencing Policy

Module name: mac_ifoff.ko

Kernel configuration line: `options MAC_IFOFF`

Boot option: `mac_ifoff_load="YES"`

The mac_ifoff(4) module is used to disable network interfaces on the fly and to keep network interfaces from being brought up during system boot. It does not use labels and does not depend on any other MAC modules.

Most of this module's control is performed through these `sysctl` tunables:

- `security.mac.ifoff.lo_enabled` enables or disables all traffic on the loopback, lo(4), interface.
- `security.mac.ifoff.bpfrecv_enabled` enables or disables all traffic on the Berkeley Packet Filter interface, bpf(4).
- `security.mac.ifoff.other_enabled` enables or disables traffic on all other interfaces.

One of the most common uses of mac_ifoff(4) is network monitoring in an environment where network traffic should not be permitted during the boot sequence. Another use would be to write a script which uses an application such as security/aide to automatically block network traffic if it finds new or altered files in protected directories.

19.5.4. The MAC Port Access Control List Policy

Module name: mac_portacl.ko

Kernel configuration line: `MAC_PORTACL`

Boot option: `mac_portacl_load="YES"`

The mac_portacl(4) module is used to limit binding to local TCP and UDP ports, making it possible to allow non-`root` users to bind to specified privileged ports below 1024.

Once loaded, this module enables the MAC policy on all sockets. The following tunables are available:

- `security.mac.portacl.enabled` enables or disables the policy completely.
- `security.mac.portacl.port_high` sets the highest port number that mac_portacl(4) protects.
- `security.mac.portacl.suser_exempt`, when set to a non-zero value, exempts the `root` user from this policy.
- `security.mac.portacl.rules` specifies the policy as a text string of the form `rule[,rule,…]`, with as many rules as needed, and where each rule is of the form `idtype:id:protocol:port`. The `idtype` is either `uid` or `gid`. The `protocol` parameter can be `tcp` or `udp`. The `port` parameter is the port number to allow the specified user or group to bind to. Only numeric values can be used for the user ID, group ID, and port parameters.

By default, ports below 1024 can only be used by privileged processes which run as `root`. For mac_portacl(4) to allow non-privileged processes to bind to ports below 1024, set the following tunables as follows:

```
# sysctl security.mac.portacl.port_high=1023
# sysctl net.inet.ip.portrange.reservedlow=0
# sysctl net.inet.ip.portrange.reservedhigh=0
```

To prevent the `root` user from being affected by this policy, set `security.mac.portacl.suser_exempt` to a non-zero value.

```
# sysctl security.mac.portacl.suser_exempt=1
```

To allow the `www` user with UID 80 to bind to port 80 without ever needing `root` privilege:

```
# sysctl security.mac.portacl.rules=uid:80:tcp:80
```

This next example permits the user with the UID of 1001 to bind to TCP ports 110 (POP3) and 995 (POP3s):

```
# sysctl security.mac.portacl.rules=uid:1001:tcp:110,uid:1001:tcp:995
```

19.5.5. The MAC Partition Policy

Module name: mac_partition.ko

Kernel configuration line: `options MAC_PARTITION`

Boot option: `mac_partition_load="YES"`

The mac_partition(4) policy drops processes into specific "partitions" based on their MAC label. Most configuration for this policy is done using setpmac(8). One `sysctl` tunable is available for this policy:

- `security.mac.partition.enabled` enables the enforcement of MAC process partitions.

When this policy is enabled, users will only be permitted to see their processes, and any others within their partition, but will not be permitted to work with utilities outside the scope of this partition. For instance, a user in the `insecure` class will not be permitted to access `top` as well as many other commands that must spawn a process.

This example adds `top` to the label set on users in the `insecure` class. All processes spawned by users in the `insecure` class will stay in the `partition/13` label.

```
# setpmac partition/13 top
```

This command displays the partition label and the process list:

```
# ps Zax
```

This command displays another user's process partition label and that user's currently running processes:

```
# ps -ZU trhodes
```

ℹ Users can see processes in `root`'s label unless the mac_seeotheruids(4) policy is loaded.

19.5.6. The MAC Multi-Level Security Module

Module name: mac_mls.ko

Kernel configuration line: `options MAC_MLS`

Boot option: `mac_mls_load="YES"`

The mac_mls(4) policy controls access between subjects and objects in the system by enforcing a strict information flow policy.

In MLS environments, a "clearance" level is set in the label of each subject or object, along with compartments. Since these clearance levels can reach numbers greater than several thousand, it would be a daunting task to thoroughly configure every subject or object. To ease this administrative overhead, three labels are included in this policy: `mls/low`, `mls/equal`, and `mls/high`, where:

- Anything labeled with `mls/low` will have a low clearance level and not be permitted to access information of a higher level. This label also prevents objects of a higher clearance level from writing or passing information to a lower level.
- `mls/equal` should be placed on objects which should be exempt from the policy.
- `mls/high` is the highest level of clearance possible. Objects assigned this label will hold dominance over all other objects in the system; however, they will not permit the leaking of information to objects of a lower class.

MLS provides:

- A hierarchical security level with a set of non-hierarchical categories.
- Fixed rules of `no read up, no write down`. This means that a subject can have read access to objects on its own level or below, but not above. Similarly, a subject can have write access to objects on its own level or above, but not beneath.
- Secrecy, or the prevention of inappropriate disclosure of data.
- A basis for the design of systems that concurrently handle data at multiple sensitivity levles

without leaking information between secret and confidential.

The following `sysctl` tunables are available:

- `security.mac.mls.enabled` is used to enable or disable the MLS policy.
- `security.mac.mls.ptys_equal` labels all pty(4) devices as `mls/equal` during creation.
- `security.mac.mls.revocation_enabled` revokes access to objects after their label changes to a label of a lower grade.
- `security.mac.mls.max_compartments` sets the maximum number of compartment levels allowed on a system.

To manipulate MLS labels, use setfmac(8). To assign a label to an object:

```
# setfmac mls/5 test
```

To get the MLS label for the file test:

```
# getfmac test
```

Another approach is to create a master policy file in /etc/ which specifies the MLS policy information and to feed that file to `setfmac`.

When using the MLS policy module, an administrator plans to control the flow of sensitive information. The default `block read up block write down` sets everything to a low state. Everything is accessible and an administrator slowly augments the confidentiality of the information.

Beyond the three basic label options, an administrator may group users and groups as required to block the information flow between them. It might be easier to look at the information in clearance levels using descriptive words, such as classifications of `Confidential`, `Secret`, and `Top Secret`. Some administrators instead create different groups based on project levels. Regardless of the classification method, a well thought out plan must exist before implementing a restrictive policy.

Some example situations for the MLS policy module include an e-commerce web server, a file server holding critical company information, and financial institution environments.

19.5.7. The MAC Biba Module

Module name: mac_biba.ko

Kernel configuration line: `options MAC_BIBA`

Boot option: `mac_biba_load="YES"`

The mac_biba(4) module loads the MAC Biba policy. This policy is similar to the MLS policy with the exception that the rules for information flow are slightly reversed. This is to prevent the downward flow of sensitive information whereas the MLS policy prevents the upward flow of sensitive information.

In Biba environments, an "integrity" label is set on each subject or object. These labels are made up of hierarchical grades and non-hierarchical components. As a grade ascends, so does its integrity.

Supported labels are `biba/low`, `biba/equal`, and `biba/high`, where:

- `biba/low` is considered the lowest integrity an object or subject may have. Setting this on objects or subjects blocks their write access to objects or subjects marked as `biba/high`, but will not prevent read access.
- `biba/equal` should only be placed on objects considered to be exempt from the policy.
- `biba/high` permits writing to objects set at a lower label, but does not permit reading that object. It is recommended that this label be placed on objects that affect the integrity of the entire system.

Biba provides:

- Hierarchical integrity levels with a set of non-hierarchical integrity categories.
- Fixed rules are `no write up, no read down`, the opposite of MLS. A subject can have write access to objects on its own level or below, but not above. Similarly, a subject can have read access to objects on its own level or above, but not below.
- Integrity by preventing inappropriate modification of data.
- Integrity levels instead of MLS sensitivity levels.

The following tunables can be used to manipulate the Biba policy:

- `security.mac.biba.enabled` is used to enable or disable enforcement of the Biba policy on the target machine.
- `security.mac.biba.ptys_equal` is used to disable the Biba policy on pty(4) devices.
- `security.mac.biba.revocation_enabled` forces the revocation of access to objects if the label is changed to dominate the subject.

To access the Biba policy setting on system objects, use `setfmac` and `getfmac`:

```
# setfmac biba/low test
# getfmac test
test: biba/low
```

Integrity, which is different from sensitivity, is used to guarantee that information is not manipulated by untrusted parties. This includes information passed between subjects and objects. It ensures that users will only be able to modify or access information they have been given explicit access to. The mac_biba(4) security policy module permits an administrator to configure which files and programs a user may see and invoke while assuring that the programs and files are trusted by the system for that user.

During the initial planning phase, an administrator must be prepared to partition users into grades, levels, and areas. The system will default to a high label once this policy module is enabled, and it is up to the administrator to configure the different grades and levels for users. Instead of using clearance levels, a good planning method could include topics. For instance, only allow developers modification access to the source code repository, source code compiler, and other development utilities. Other users would be grouped into other categories such as testers, designers, or end users and would only be permitted read access.

A lower integrity subject is unable to write to a higher integrity subject and a higher integrity subject cannot list or read a lower integrity object. Setting a label at the lowest possible grade could make it inaccessible to subjects. Some prospective environments for this security policy module would include a constrained web server, a development and test machine, and a source code repository. A less useful implementation would be a personal workstation, a machine used as a router, or a network firewall.

19.5.8. The MAC Low-watermark Module

Module name: mac_lomac.ko

Kernel configuration line: `options MAC_LOMAC`

Boot option: `mac_lomac_load="YES"`

Unlike the MAC Biba policy, the mac_lomac(4) policy permits access to lower integrity objects only after decreasing the integrity level to not disrupt any integrity rules.

The Low-watermark integrity policy works almost identically to Biba, with the exception of using floating labels to support subject demotion via an auxiliary grade compartment. This secondary compartment takes the form `[auxgrade]`. When assigning a policy with an auxiliary grade, use the syntax `lomac/10[2]`, where `2` is the auxiliary grade.

This policy relies on the ubiquitous labeling of all system objects with integrity labels, permitting subjects to read from low integrity objects and then downgrading the label on the subject to prevent future writes to high integrity objects using `[auxgrade]`. The policy may provide greater compatibility and require less initial configuration than Biba.

Like the Biba and MLS policies, `setfmac` and `setpmac` are used to place labels on system objects:

```
# setfmac /usr/home/trhodes lomac/high[low]
# getfmac /usr/home/trhodes lomac/high[low]
```

The auxiliary grade `low` is a feature provided only by the MACLOMAC policy.

19.6. User Lock Down

This example considers a relatively small storage system with fewer than fifty users. Users will have login capabilities and are permitted to store data and access resources.

For this scenario, the mac_bsdextended(4) and mac_seeotheruids(4) policy modules could co-exist and block access to system objects while hiding user processes.

Begin by adding the following line to /boot/loader.conf:

```
mac_seeotheruids_load="YES"
```

The mac_bsdextended(4) security policy module may be activated by adding this line to /etc/rc.conf:

```
ugidfw_enable="YES"
```

Default rules stored in /etc/rc.bsdextended will be loaded at system initialization. However, the default entries may need modification. Since this machine is expected only to service users, everything may be left commented out except the last two lines in order to force the loading of user owned system objects by default.

Add the required users to this machine and reboot. For testing purposes, try logging in as a different user across two consoles. Run `ps aux` to see if processes of other users are visible. Verify that running ls(1) on another user's home directory fails.

Do not try to test with the `root` user unless the specific `sysctl`s have been modified to block super user access.

When a new user is added, their mac_bsdextended(4) rule will not be in the ruleset list. To update the ruleset quickly, unload the security policy module and reload it again using kldunload(8) and kldload(8).

19.7. Nagios in a MAC Jail

This section demonstrates the steps that are needed to implement the Nagios network monitoring system in a MAC environment. This is meant as an example which still requires the administrator to test that the implemented policy meets the security requirements of the network before using in a production environment.

This example requires `multilabel` to be set on each file system. It also assumes that net-mgmt/nagios-plugins, net-mgmt/nagios, and www/apache22 are all installed, configured, and working correctly before attempting the integration into the MAC framework.

19.7.1. Create an Insecure User Class

Begin the procedure by adding the following user class to /etc/login.conf:

```
insecure:\
:welcome=/etc/motd:\
:setenv=MAIL=/var/mail/$,BLOCKSIZE=K:\
:path=~/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin
:manpath=/usr/share/man /usr/local/man:\
:nologin=/usr/sbin/nologin:\
:cputime=1h30m:\
:datasize=8M:\
:vmemoryuse=100M:\
:stacksize=2M:\
:memorylocked=4M:\
:memoryuse=8M:\
:filesize=8M:\
:coredumpsize=8M:\
:openfiles=24:\
:maxproc=32:\
:priority=0:\
:requirehome:\
:passwordtime=91d:\
:umask=022:\
:ignoretime@:\
:label=biba/10(10-10):
```

Then, add the following line to the default user class section:

```
:label=biba/high:
```

Save the edits and issue the following command to rebuild the database:

```
# cap_mkdb /etc/login.conf
```

19.7.2. Configure Users

Set the `root` user to the default class using:

```
# pw usermod root -L default
```

All user accounts that are not `root` will now require a login class. The login class is required, otherwise users will be refused access to common commands. The following `sh` script should do the trick:

```
# for x in `awk -F: '($3 >= 1001) && ($3 != 65534) { print $1 }' \
    /etc/passwd`; do pw usermod $x -L default; done;
```

Next, drop the `nagios` and `www` accounts into the insecure class:

```
# pw usermod nagios -L insecure
# pw usermod www -L insecure
```

19.7.3. Create the Contexts File

A contexts file should now be created as /etc/policy.contexts:

```
# This is the default BIBA policy for this system.

# System:
/var/run(/.*)?          biba/equal

/dev/(/.*)?         biba/equal

/var                biba/equal
/var/spool(/.*)?        biba/equal

/var/log(/.*)?          biba/equal

/tmp(/.*)?          biba/equal
/var/tmp(/.*)?          biba/equal

/var/spool/mqueue       biba/equal
/var/spool/clientmqueue     biba/equal

# For Nagios:
/usr/local/etc/nagios(/.*)? biba/10

/var/spool/nagios(/.*)?     biba/10

# For apache
/usr/local/etc/apache(/.*)? biba/10
```

This policy enforces security by setting restrictions on the flow of information. In this specific configuration, users, including `root`, should never be allowed to access Nagios. Configuration files and processes that are a part of Nagios will be completely self contained or jailed.

This file will be read after running `setfsmac` on every file system. This example sets the policy on the root file system:

```
# setfsmac -ef /etc/policy.contexts /
```

Next, add these edits to the main section of /etc/mac.conf:

```
default_labels file ?biba
default_labels ifnet ?biba
default_labels process ?biba
default_labels socket ?biba
```

19.7.4. Loader Configuration

To finish the configuration, add the following lines to /boot/loader.conf:

```
mac_biba_load="YES"
mac_seeotheruids_load="YES"
security.mac.biba.trust_all_interfaces=1
```

And the following line to the network card configuration stored in /etc/rc.conf. If the primary network configuration is done via DHCP, this may need to be configured manually after every system boot:

```
maclabel biba/equal
```

19.7.5. Testing the Configuration

First, ensure that the web server and Nagios will not be started on system initialization and reboot. Ensure that `root` cannot access any of the files in the Nagios configuration directory. If `root` can list the contents of /var/spool/nagios, something is wrong. Instead, a "permission denied" error should be returned.

If all seems well, Nagios, Apache, and Sendmail can now be started:

```
# cd /etc/mail && make stop && \
setpmac biba/equal make start && setpmac biba/10\(10-10\) apachectl start && \
setpmac biba/10\(10-10\) /usr/local/etc/rc.d/nagios.sh forcestart
```

Double check to ensure that everything is working properly. If not, check the log files for error messages. If needed, use sysctl(8) to disable the mac_biba(4) security policy module and try starting everything again as usual.

> The `root` user can still change the security enforcement and edit its configuration files. The following command will permit the degradation of the security policy to a lower grade for a newly spawned shell:
>
> ```
> # setpmac biba/10 csh
> ```
>
> To block this from happening, force the user into a range using login.conf(5). If setpmac(8) attempts to run a command outside of the compartment's range, an error will be returned and the command will not be executed. In this case, set root to `biba/high(high-high)`.

19.8. Troubleshooting the MAC Framework

This section discusses common configuration errors and how to resolve them.

The `multilabel` flag does not stay enabled on the root (/) partition

The following steps may resolve this transient error:

1. Edit /etc/fstab and set the root partition to `ro` for read-only.
2. Reboot into single user mode.
3. Run `tunefs -l enable` on /.
4. Reboot the system.
5. Run `mount -urw`/ and change the `ro` back to `rw` in /etc/fstab and reboot the system again.
6. Double-check the output from `mount` to ensure that `multilabel` has been properly set on the root file system.

After establishing a secure environment with MAC, Xorg no longer starts

This could be caused by the MAC `partition` policy or by a mislabeling in one of the MAC labeling policies. To debug, try the following:

1. Check the error message. If the user is in the `insecure` class, the `partition` policy may be the culprit. Try setting the user's class back to the `default` class and rebuild the database with `cap_mkdb`. If this does not alleviate the problem, go to step two.
2. Double-check that the label policies are set correctly for the user, Xorg, and the /dev entries.
3. If neither of these resolve the problem, send the error message and a description of the environment to the FreeBSD general questions mailing list.

The `_secure_path: unable to stat .login_conf` error appears

This error can appear when a user attempts to switch from the `root` user to another user in the system. This message usually occurs when the user has a higher label setting than that of the user they are attempting to become. For instance, if `joe` has a default label of `biba/low` and `root` has a label of `biba/high`, `root` cannot view `joe`'s home directory. This will happen whether or not `root` has used `su` to become `joe` as the Biba integrity model will not permit `root` to view objects set at a lower integrity level.

The system no longer recognizes `root`

When this occurs, `whoami` returns `0` and `su` returns `who are you?`.

This can happen if a labeling policy has been disabled by sysctl(8) or the policy module was unloaded. If the policy is disabled, the login capabilities database needs to be reconfigured. Double check /etc/login.conf to ensure that all `label` options have been removed and rebuild the database with `cap_mkdb`.

This may also happen if a policy restricts access to master.passwd. This is usually caused by an administrator altering the file under a label which conflicts with the general policy being used by the system. In these cases, the user information would be read by the system and access would be blocked as the file has inherited the new label. Disable the policy using sysctl(8) and everything should return to normal.

Chapter 20. Security Event Auditing

20.1. Synopsis

The FreeBSD operating system includes support for security event auditing. Event auditing supports reliable, fine-grained, and configurable logging of a variety of security-relevant system events, including logins, configuration changes, and file and network access. These log records can be invaluable for live system monitoring, intrusion detection, and postmortem analysis. FreeBSD implements Sun™'s published Basic Security Module (BSM) Application Programming Interface (API) and file format, and is interoperable with the Solaris™ and Mac OS® X audit implementations.

This chapter focuses on the installation and configuration of event auditing. It explains audit policies and provides an example audit configuration.

Read this chapter to learn:

- What event auditing is and how it works.
- How to configure event auditing on FreeBSD for users and processes.
- How to review the audit trail using the audit reduction and review tools.

Before reading this chapter:

- Understand UNIX® and FreeBSD basics (FreeBSD Basics).
- Be familiar with the basics of kernel configuration/compilation (Configuring the FreeBSD Kernel).
- Have some familiarity with security and how it pertains to FreeBSD (Security).

⚠

The audit facility has some known limitations. Not all security-relevant system events are auditable and some login mechanisms, such as Xorg-based display managers and third-party daemons, do not properly configure auditing for user login sessions.

The security event auditing facility is able to generate very detailed logs of system activity. On a busy system, trail file data can be very large when configured for high detail, exceeding gigabytes a week in some configurations. Administrators should take into account the disk space requirements associated with high volume audit configurations. For example, it may be desirable to dedicate a file system to /var/audit so that other file systems are not affected if the audit file system becomes full.

20.2. Key Terms

The following terms are related to security event auditing:

- *event*: an auditable event is any event that can be logged using the audit subsystem. Examples of security-relevant events include the creation of a file, the building of a network connection, or a user logging in. Events are either "attributable", meaning that they can be traced to an authenticated user, or "non-attributable". Examples of non-attributable events are any events that occur before authentication in the login process, such as bad password attempts.

- *class*: a named set of related events which are used in selection expressions. Commonly used classes of events include "file creation" (fc), "exec" (ex), and "login_logout" (lo).
- *record*: an audit log entry describing a security event. Records contain a record event type, information on the subject (user) performing the action, date and time information, information on any objects or arguments, and a success or failure condition.
- *trail*: a log file consisting of a series of audit records describing security events. Trails are in roughly chronological order with respect to the time events completed. Only authorized processes are allowed to commit records to the audit trail.
- *selection expression*: a string containing a list of prefixes and audit event class names used to match events.
- *preselection*: the process by which the system identifies which events are of interest to the administrator. The preselection configuration uses a series of selection expressions to identify which classes of events to audit for which users, as well as global settings that apply to both authenticated and unauthenticated processes.
- *reduction*: the process by which records from existing audit trails are selected for preservation, printing, or analysis. Likewise, the process by which undesired audit records are removed from the audit trail. Using reduction, administrators can implement policies for the preservation of audit data. For example, detailed audit trails might be kept for one month, but after that, trails might be reduced in order to preserve only login information for archival purposes.

20.3. Audit Configuration

User space support for event auditing is installed as part of the base FreeBSD operating system. Kernel support is available in the GENERIC kernel by default, and auditd(8) can be enabled by adding the following line to /etc/rc.conf:

```
auditd_enable="YES"
```

Then, start the audit daemon:

```
# service auditd start
```

Users who prefer to compile a custom kernel must include the following line in their custom kernel configuration file:

```
options AUDIT
```

20.3.1. Event Selection Expressions

Selection expressions are used in a number of places in the audit configuration to determine which events should be audited. Expressions contain a list of event classes to match. Selection expressions are evaluated from left to right, and two expressions are combined by appending one onto the other.

Default Audit Event Classes summarizes the default audit event classes:

Table 32. Default Audit Event Classes

Class Name	Description	Action
all	all	Match all event classes.
aa	authentication and authorization	
ad	administrative	Administrative actions performed on the system as a whole.
ap	application	Application defined action.
cl	file close	Audit calls to the `close` system call.
ex	exec	Audit program execution. Auditing of command line arguments and environmental variables is controlled via audit_control(5) using the `argv` and `envv` parameters to the `policy` setting.
fa	file attribute access	Audit the access of object attributes such as stat(1) and pathconf(2).
fc	file create	Audit events where a file is created as a result.
fd	file delete	Audit events where file deletion occurs.
fm	file attribute modify	Audit events where file attribute modification occurs, such as by chown(8), chflags(1), and flock(2).
fr	file read	Audit events in which data is read or files are opened for reading.

Class Name	Description	Action
fw	file write	Audit events in which data is written or files are written or modified.
io	ioctl	Audit use of the `ioctl` system call.
ip	ipc	Audit various forms of Inter-Process Communication, including POSIX pipes and System V IPC operations.
lo	login_logout	Audit login(1) and logout(1) events.
na	non attributable	Audit non-attributable events.
no	invalid class	Match no audit events.
nt	network	Audit events related to network actions such as connect(2) and accept(2).
ot	other	Audit miscellaneous events.
pc	process	Audit process operations such as exec(3) and exit(3).

These audit event classes may be customized by modifying the audit_class and audit_event configuration files.

Each audit event class may be combined with a prefix indicating whether successful/failed operations are matched, and whether the entry is adding or removing matching for the class and type. Prefixes for Audit Event Classes summarizes the available prefixes:

Table 33. Prefixes for Audit Event Classes

Prefix	Action
+	Audit successful events in this class.
-	Audit failed events in this class.
^	Audit neither successful nor failed events in this class.
^+	Do not audit successful events in this class.
^-	Do not audit failed events in this class.

If no prefix is present, both successful and failed instances of the event will be audited.

The following example selection string selects both successful and failed login/logout events, but only successful execution events:

```
lo,+ex
```

20.3.2. Configuration Files

The following configuration files for security event auditing are found in /etc/security:

- audit_class: contains the definitions of the audit classes.
- audit_control: controls aspects of the audit subsystem, such as default audit classes, minimum disk space to leave on the audit log volume, and maximum audit trail size.
- audit_event: textual names and descriptions of system audit events and a list of which classes each event is in.
- audit_user: user-specific audit requirements to be combined with the global defaults at login.
- audit_warn: a customizable shell script used by auditd(8) to generate warning messages in exceptional situations, such as when space for audit records is running low or when the audit trail file has been rotated.

> Audit configuration files should be edited and maintained carefully, as errors in configuration may result in improper logging of events.

In most cases, administrators will only need to modify audit_control and audit_user. The first file controls system-wide audit properties and policies and the second file may be used to fine-tune auditing by user.

The audit_control File

A number of defaults for the audit subsystem are specified in audit_control:

```
dir:/var/audit
dist:off
flags:lo,aa
minfree:5
naflags:lo,aa
policy:cnt,argv
filesz:2M
expire-after:10M
```

The `dir` entry is used to set one or more directories where audit logs will be stored. If more than one directory entry appears, they will be used in order as they fill. It is common to configure audit so that audit logs are stored on a dedicated file system, in order to prevent interference between the audit subsystem and other subsystems if the file system fills.

If the `dist` field is set to `on` or `yes`, hard links will be created to all trail files in /var/audit/dist.

The `flags` field sets the system-wide default preselection mask for attributable events. In the example above, successful and failed login/logout events as well as authentication and authorization are audited for all users.

The `minfree` entry defines the minimum percentage of free space for the file system where the audit trail is stored.

The `naflags` entry specifies audit classes to be audited for non-attributed events, such as the login/logout process and authentication and authorization.

The `policy` entry specifies a comma-separated list of policy flags controlling various aspects of audit behavior. The `cnt` indicates that the system should continue running despite an auditing failure (this flag is highly recommended). The other flag, `argv`, causes command line arguments to the execve(2) system call to be audited as part of command execution.

The `filesz` entry specifies the maximum size for an audit trail before automatically terminating and rotating the trail file. A value of `0` disables automatic log rotation. If the requested file size is below the minimum of 512k, it will be ignored and a log message will be generated.

The `expire-after` field specifies when audit log files will expire and be removed.

The audit_user File

The administrator can specify further audit requirements for specific users in audit_user. Each line configures auditing for a user via two fields: the `alwaysaudit` field specifies a set of events that should always be audited for the user, and the `neveraudit` field specifies a set of events that should never be audited for the user.

The following example entries audit login/logout events and successful command execution for `root` and file creation and successful command execution for `www`. If used with the default audit_control, the `lo` entry for `root` is redundant, and login/logout events will also be audited for `www`.

```
root:lo,+ex:no
www:fc,+ex:no
```

20.4. Working with Audit Trails

Since audit trails are stored in the BSM binary format, several built-in tools are available to modify or convert these trails to text. To convert trail files to a simple text format, use `praudit`. To reduce the audit trail file for analysis, archiving, or printing purposes, use `auditreduce`. This utility supports a variety of selection parameters, including event type, event class, user, date or time of the event, and the file path or object acted on.

For example, to dump the entire contents of a specified audit log in plain text:

```
# praudit /var/audit/AUDITFILE
```

Where *AUDITFILE* is the audit log to dump.

Audit trails consist of a series of audit records made up of tokens, which `praudit` prints sequentially, one per line. Each token is of a specific type, such as `header` (an audit record header) or `path` (a file path from a name lookup). The following is an example of an `execve` event:

```
header,133,10,execve(2),0,Mon Sep 25 15:58:03 2006, + 384 msec
exec arg,finger,doug
path,/usr/bin/finger
attribute,555,root,wheel,90,24918,104944
subject,robert,root,wheel,root,wheel,38439,38032,42086,128.232.9.100
return,success,0
trailer,133
```

This audit represents a successful `execve` call, in which the command `finger doug` has been run. The `exec arg` token contains the processed command line presented by the shell to the kernel. The `path` token holds the path to the executable as looked up by the kernel. The `attribute` token describes the binary and includes the file mode. The `subject` token stores the audit user ID, effective user ID and group ID, real user ID and group ID, process ID, session ID, port ID, and login address. Notice that the audit user ID and real user ID differ as the user `robert` switched to the `root` account before running this command, but it is audited using the original authenticated user. The `return` token indicates the successful execution and the `trailer` concludes the record.

XML output format is also supported and can be selected by including `-x`.

Since audit logs may be very large, a subset of records can be selected using `auditreduce`. This example selects all audit records produced for the user `trhodes` stored in AUDITFILE:

```
# auditreduce -u trhodes /var/audit/AUDITFILE | praudit
```

Members of the `audit` group have permission to read audit trails in /var/audit. By default, this group is empty, so only the `root` user can read audit trails. Users may be added to the `audit` group in order to delegate audit review rights. As the ability to track audit log contents provides significant insight into the behavior of users and processes, it is recommended that the delegation of audit review rights be performed with caution.

20.4.1. Live Monitoring Using Audit Pipes

Audit pipes are cloning pseudo-devices which allow applications to tap the live audit record stream. This is primarily of interest to authors of intrusion detection and system monitoring applications. However, the audit pipe device is a convenient way for the administrator to allow live monitoring without running into problems with audit trail file ownership or log rotation interrupting the event stream. To track the live audit event stream:

```
# praudit /dev/auditpipe
```

By default, audit pipe device nodes are accessible only to the `root` user. To make them accessible to the members of the `audit` group, add a `devfs` rule to /etc/devfs.rules:

```
add path 'auditpipe*' mode 0440 group audit
```

See devfs.rules(5) for more information on configuring the devfs file system.

It is easy to produce audit event feedback cycles, in which the viewing of each audit event results in the generation of more audit events. For example, if all network I/O is audited, and `praudit` is run from an SSH session, a continuous stream of audit events will be generated at a high rate, as each event being printed will generate another event. For this reason, it is advisable to run `praudit` on an audit pipe device from sessions without fine-grained I/O auditing.

20.4.2. Rotating and Compressing Audit Trail Files

Audit trails are written to by the kernel and managed by the audit daemon, auditd(8). Administrators should not attempt to use newsyslog.conf(5) or other tools to directly rotate audit logs. Instead, `audit` should be used to shut down auditing, reconfigure the audit system, and perform log rotation. The following command causes the audit daemon to create a new audit log and signal the kernel to switch to using the new log. The old log will be terminated and renamed, at which point it may then be manipulated by the administrator:

```
# audit -n
```

If auditd(8) is not currently running, this command will fail and an error message will be produced.

Adding the following line to /etc/crontab will schedule this rotation every twelve hours:

```
0     */12       *       *       *       root    /usr/sbin/audit -n
```

The change will take effect once /etc/crontab is saved.

Automatic rotation of the audit trail file based on file size is possible using `filesz` in audit_control as described in The audit_control File.

As audit trail files can become very large, it is often desirable to compress or otherwise archive trails once they have been closed by the audit daemon. The audit_warn script can be used to perform customized operations for a variety of audit-related events, including the clean termination of audit trails when they are rotated. For example, the following may be added to /etc/security/audit_warn to compress audit trails on close:

```
#
# Compress audit trail files on close.
#
if [ "$1" = closefile ]; then
        gzip -9 $2
fi
```

Other archiving activities might include copying trail files to a centralized server, deleting old trail files, or reducing the audit trail to remove unneeded records. This script will be run only when audit trail files are cleanly terminated. It will not be run on trails left unterminated following an improper shutdown.

Chapter 21. Storage

21.1. Synopsis

This chapter covers the use of disks and storage media in FreeBSD. This includes SCSI and IDE disks, CD and DVD media, memory-backed disks, and USB storage devices.

Read this chapter to learn:

- How to add additional hard disks to a FreeBSD system.
- How to grow the size of a disk's partition on FreeBSD.
- How to configure FreeBSD to use USB storage devices.
- How to use CD and DVD media on a FreeBSD system.
- How to use the backup programs available under FreeBSD.
- How to set up memory disks.
- What file system snapshots are and how to use them efficiently.
- How to use quotas to limit disk space usage.
- How to encrypt disks and swap to secure them against attackers.
- How to configure a highly available storage network.

Before reading this chapter:

- Know how to configure and install a new FreeBSD kernel.

21.2. Adding Disks

This section describes how to add a new SATA disk to a machine that currently only has a single drive. First, turn off the computer and install the drive in the computer following the instructions of the computer, controller, and drive manufacturers. Reboot the system and become `root`.

Inspect /var/run/dmesg.boot to ensure the new disk was found. In this example, the newly added SATA drive will appear as ada1.

For this example, a single large partition will be created on the new disk. The GPT partitioning scheme will be used in preference to the older and less versatile MBR scheme.

If the disk to be added is not blank, old partition information can be removed with `gpart delete`. See gpart(8) for details.

The partition scheme is created, and then a single partition is added. To improve performance on newer disks with larger hardware block sizes, the partition is aligned to one megabyte boundaries:

```
# gpart create -s GPT ada1
# gpart add -t freebsd-ufs -a 1M ada1
```

Depending on use, several smaller partitions may be desired. See gpart(8) for options to create partitions smaller than a whole disk.

The disk partition information can be viewed with `gpart show`:

```
% gpart show ada1
=>        34  1465146988  ada1  GPT  (699G)
          34        2014        - free -  (1.0M)
        2048  1465143296     1  freebsd-ufs  (699G)
  1465145344        1678        - free -  (839K)
```

A file system is created in the new partition on the new disk:

```
# newfs -U /dev/ada1p1
```

An empty directory is created as a *mountpoint*, a location for mounting the new disk in the original disk's file system:

```
# mkdir /newdisk
```

Finally, an entry is added to /etc/fstab so the new disk will be mounted automatically at startup:

```
/dev/ada1p1 /newdisk    ufs rw  2   2
```

The new disk can be mounted manually, without restarting the system:

```
# mount /newdisk
```

21.3. Resizing and Growing Disks

A disk's capacity can increase without any changes to the data already present. This happens commonly with virtual machines, when the virtual disk turns out to be too small and is enlarged. Sometimes a disk image is written to a USB memory stick, but does not use the full capacity. Here we describe how to resize or *grow* disk contents to take advantage of increased capacity.

Determine the device name of the disk to be resized by inspecting /var/run/dmesg.boot. In this example, there is only one SATA disk in the system, so the drive will appear as ada0.

List the partitions on the disk to see the current configuration:

```
# gpart show ada0
```

```
=>       34  83886013  ada0  GPT  (48G) [CORRUPT]
         34       128     1  freebsd-boot  (64k)
        162  79691648     2  freebsd-ufs  (38G)
   79691810   4194236     3  freebsd-swap  (2G)
   83886046         1        - free -  (512B)
```

If the disk was formatted with the GPT partitioning scheme, it may show as "corrupted" because the GPT backup partition table is no longer at the end of the drive. Fix the backup partition table with `gpart`:

```
# gpart recover ada0
ada0 recovered
```

Now the additional space on the disk is available for use by a new partition, or an existing partition can be expanded:

```
# gpart show ada0
=>       34  102399933  ada0  GPT  (48G)
         34        128     1  freebsd-boot  (64k)
        162   79691648     2  freebsd-ufs  (38G)
   79691810    4194236     3  freebsd-swap  (2G)
   83886046   18513921        - free -  (8.8G)
```

Partitions can only be resized into contiguous free space. Here, the last partition on the disk is the swap partition, but the second partition is the one that needs to be resized. Swap partitions only contain temporary data, so it can safely be unmounted, deleted, and then recreate the third partition after resizing the second partition.

Disable the swap partition:

```
# swapoff /dev/ada0p3
```

Delete the third partition, specified by the `-i` flag, from the disk *ada0*.

```
# gpart delete -i 3 ada0
ada0p3 deleted
# gpart show ada0
=>       34  102399933  ada0  GPT  (48G)
         34        128     1  freebsd-boot  (64k)
        162   79691648     2  freebsd-ufs  (38G)
   79691810   22708157        - free -  (10G)
```

There is risk of data loss when modifying the partition table of a mounted file system. It is best to perform the following steps on an unmounted file system while

> running from a live CD-ROM or USB device. However, if absolutely necessary, a mounted file system can be resized after disabling GEOM safety features:
>
> ```
> # sysctl kern.geom.debugflags=16
> ```

Resize the partition, leaving room to recreate a swap partition of the desired size. The partition to resize is specified with `-i`, and the new desired size with `-s`. Optionally, alignment of the partition is controlled with `-a`. This only modifies the size of the partition. The file system in the partition will be expanded in a separate step.

```
# gpart resize -i 2 -s 47G -a 4k ada0
ada0p2 resized
# gpart show ada0
=>        34  102399933  ada0  GPT  (48G)
          34        128     1  freebsd-boot  (64k)
         162   98566144     2  freebsd-ufs  (47G)
    98566306    3833661        - free -  (1.8G)
```

Recreate the swap partition and activate it. If no size is specified with `-s`, all remaining space is used:

```
# gpart add -t freebsd-swap -a 4k ada0
ada0p3 added
# gpart show ada0
=>        34  102399933  ada0  GPT  (48G)
          34        128     1  freebsd-boot  (64k)
         162   98566144     2  freebsd-ufs  (47G)
    98566306    3833661     3  freebsd-swap  (1.8G)
# swapon /dev/ada0p3
```

Grow the UFS file system to use the new capacity of the resized partition:

```
# growfs /dev/ada0p2
Device is mounted read-write; resizing will result in temporary write suspension for
/.
It's strongly recommended to make a backup before growing the file system.
OK to grow file system on /dev/ada0p2, mounted on /, from 38GB to 47GB? [Yes/No] Yes
super-block backups (for fsck -b #) at:
 80781312, 82063552, 83345792, 84628032, 85910272, 87192512, 88474752,
 89756992, 91039232, 92321472, 93603712, 94885952, 96168192, 97450432
```

If the file system is ZFS, the resize is triggered by running the `online` subcommand with `-e`:

```
# zpool online -e zroot /dev/ada0p2
```

Both the partition and the file system on it have now been resized to use the newly-available disk space.

21.4. USB Storage Devices

Many external storage solutions, such as hard drives, USB thumbdrives, and CD and DVD burners, use the Universal Serial Bus (USB). FreeBSD provides support for USB 1.x, 2.0, and 3.0 devices.

> USB 3.0 support is not compatible with some hardware, including Haswell (Lynx point) chipsets. If FreeBSD boots with a `failed with error 19` message, disable xHCI/USB3 in the system BIOS.

Support for USB storage devices is built into the GENERIC kernel. For a custom kernel, be sure that the following lines are present in the kernel configuration file:

```
device scbus    # SCSI bus (required for ATA/SCSI)
device da   # Direct Access (disks)
device pass # Passthrough device (direct ATA/SCSI access)
device uhci # provides USB 1.x support
device ohci # provides USB 1.x support
device ehci # provides USB 2.0 support
device xhci # provides USB 3.0 support
device usb  # USB Bus (required)
device umass    # Disks/Mass storage - Requires scbus and da
device cd   # needed for CD and DVD burners
```

FreeBSD uses the umass(4) driver which uses the SCSI subsystem to access USB storage devices. Since any USB device will be seen as a SCSI device by the system, if the USB device is a CD or DVD burner, do *not* include `device atapicam` in a custom kernel configuration file.

The rest of this section demonstrates how to verify that a USB storage device is recognized by FreeBSD and how to configure the device so that it can be used.

21.4.1. Device Configuration

To test the USB configuration, plug in the USB device. Use `dmesg` to confirm that the drive appears in the system message buffer. It should look something like this:

```
umass0: <STECH Simple Drive, class 0/0, rev 2.00/1.04, addr 3> on usbus0
umass0:  SCSI over Bulk-Only; quirks = 0x0100
umass0:4:0:-1: Attached to scbus4
da0 at umass-sim0 bus 0 scbus4 target 0 lun 0
da0: <STECH Simple Drive 1.04> Fixed Direct Access SCSI-4 device
da0: Serial Number WD-WXE508CAN263
da0: 40.000MB/s transfers
da0: 152627MB (312581808 512 byte sectors: 255H 63S/T 19457C)
da0: quirks=0x2<NO_6_BYTE>
```

The brand, device node (da0), speed, and size will differ according to the device.

Since the USB device is seen as a SCSI one, `camcontrol` can be used to list the USB storage devices attached to the system:

```
# camcontrol devlist
<STECH Simple Drive 1.04>          at scbus4 target 0 lun 0 (pass3,da0)
```

Alternately, `usbconfig` can be used to list the device. Refer to usbconfig(8) for more information about this command.

```
# usbconfig
ugen0.3: <Simple Drive STECH> at usbus0, cfg=0 md=HOST spd=HIGH (480Mbps) pwr=ON (2mA)
```

If the device has not been formatted, refer to Adding Disks for instructions on how to format and create partitions on the USB drive. If the drive comes with a file system, it can be mounted by `root` using the instructions in "Mounting and Unmounting File Systems".

> Allowing untrusted users to mount arbitrary media, by enabling `vfs.usermount` as described below, should not be considered safe from a security point of view. Most file systems were not built to safeguard against malicious devices.

To make the device mountable as a normal user, one solution is to make all users of the device a member of the `operator` group using pw(8). Next, ensure that `operator` is able to read and write the device by adding these lines to /etc/devfs.rules:

```
[localrules=5]
add path 'da*' mode 0660 group operator
```

> If internal SCSI disks are also installed in the system, change the second line as follows:
>
> ```
> add path 'da[3-9]*' mode 0660 group operator
> ```
>
> This will exclude the first three SCSI disks (da0 to da2) from belonging to the `operator` group. Replace *3* with the number of internal SCSI disks. Refer to devfs.rules(5) for more information about this file.

Next, enable the ruleset in /etc/rc.conf:

```
devfs_system_ruleset="localrules"
```

Then, instruct the system to allow regular users to mount file systems by adding the following line to /etc/sysctl.conf:

```
vfs.usermount=1
```

Since this only takes effect after the next reboot, use `sysctl` to set this variable now:

```
# sysctl vfs.usermount=1
vfs.usermount: 0 -> 1
```

The final step is to create a directory where the file system is to be mounted. This directory needs to be owned by the user that is to mount the file system. One way to do that is for `root` to create a subdirectory owned by that user as /mnt/username. In the following example, replace *username* with the login name of the user and *usergroup* with the user's primary group:

```
# mkdir /mnt/username
# chown username:usergroup /mnt/username
```

Suppose a USB thumbdrive is plugged in, and a device /dev/da0s1 appears. If the device is formatted with a FAT file system, the user can mount it using:

```
% mount -t msdosfs -o -m=644,-M=755 /dev/da0s1 /mnt/username
```

Before the device can be unplugged, it *must* be unmounted first:

```
% umount /mnt/username
```

After device removal, the system message buffer will show messages similar to the following:

```
umass0: at uhub3, port 2, addr 3 (disconnected)
da0 at umass-sim0 bus 0 scbus4 target 0 lun 0
da0: <STECH Simple Drive 1.04> s/n WD-WXE508CAN263          detached
(da0:umass-sim0:0:0:0): Periph destroyed
```

21.4.2. Automounting Removable Media

USB devices can be automatically mounted by uncommenting this line in /etc/auto_master:

```
/media      -media      -nosuid
```

Then add these lines to /etc/devd.conf:

```
notify 100 {
    match "system" "GEOM";
    match "subsystem" "DEV";
    action "/usr/sbin/automount -c";
};
```

Reload the configuration if autofs(5) and devd(8) are already running:

```
# service automount restart
# service devd restart
```

autofs(5) can be set to start at boot by adding this line to /etc/rc.conf:

```
autofs_enable="YES"
```

autofs(5) requires devd(8) to be enabled, as it is by default.

Start the services immediately with:

```
# service automount start
# service automountd start
# service autounmountd start
# service devd start
```

Each file system that can be automatically mounted appears as a directory in /media/. The directory is named after the file system label. If the label is missing, the directory is named after the device node.

The file system is transparently mounted on the first access, and unmounted after a period of inactivity. Automounted drives can also be unmounted manually:

```
# automount -fu
```

This mechanism is typically used for memory cards and USB memory sticks. It can be used with any block device, including optical drives or iSCSILUNs.

21.5. Creating and Using CD Media

Compact Disc (CD) media provide a number of features that differentiate them from conventional disks. They are designed so that they can be read continuously without delays to move the head between tracks. While CD media do have tracks, these refer to a section of data to be read continuously, and not a physical property of the disk. The ISO 9660 file system was designed to deal with these differences.

The FreeBSD Ports Collection provides several utilities for burning and duplicating audio and data CDs. This chapter demonstrates the use of several command line utilities. For CD burning software with a graphical utility, consider installing the sysutils/xcdroast or sysutils/k3b packages or ports.

21.5.1. Supported Devices

The GENERIC kernel provides support for SCSI, USB, and ATAPICD readers and burners. If a custom kernel is used, the options that need to be present in the kernel configuration file vary by the type of device.

For a SCSI burner, make sure these options are present:

```
device scbus    # SCSI bus (required for ATA/SCSI)
device da   # Direct Access (disks)
device pass # Passthrough device (direct ATA/SCSI access)
device cd   # needed for CD and DVD burners
```

For a USB burner, make sure these options are present:

```
device scbus    # SCSI bus (required for ATA/SCSI)
device da   # Direct Access (disks)
device pass # Passthrough device (direct ATA/SCSI access)
device cd   # needed for CD and DVD burners
device uhci # provides USB 1.x support
device ohci # provides USB 1.x support
device ehci # provides USB 2.0 support
device xhci # provides USB 3.0 support
device usb  # USB Bus (required)
device umass    # Disks/Mass storage - Requires scbus and da
```

For an ATAPI burner, make sure these options are present:

```
device ata  # Legacy ATA/SATA controllers
device scbus    # SCSI bus (required for ATA/SCSI)
device pass # Passthrough device (direct ATA/SCSI access)
device cd   # needed for CD and DVD burners
```

On FreeBSD versions prior to 10.x, this line is also needed in the kernel configuration file if the burner is an ATAPI device:

```
device atapicam
```

Alternately, this driver can be loaded at boot time by adding the following line to /boot/loader.conf:

```
atapicam_load="YES"
```

This will require a reboot of the system as this driver can only be loaded at boot time.

To verify that FreeBSD recognizes the device, run `dmesg` and look for an entry for the device. On systems prior to 10.x, the device name in the first line of the output will be acd0 instead of cd0.

```
% dmesg | grep cd
cd0 at ahcich1 bus 0 scbus1 target 0 lun 0
cd0: <HL-DT-ST DVDRAM GU70N LT20> Removable CD-ROM SCSI-0 device
cd0: Serial Number M3OD3S34152
cd0: 150.000MB/s transfers (SATA 1.x, UDMA6, ATAPI 12bytes, PIO 8192bytes)
cd0: Attempt to query device size failed: NOT READY, Medium not present - tray closed
```

21.5.2. Burning a CD

In FreeBSD, `cdrecord` can be used to burn CDs. This command is installed with the sysutils/cdrtools package or port.

While `cdrecord` has many options, basic usage is simple. Specify the name of the ISO file to burn and, if the system has multiple burner devices, specify the name of the device to use:

```
# cdrecord dev=device imagefile.iso
```

To determine the device name of the burner, use `-scanbus` which might produce results like this:

```
# cdrecord -scanbus
ProDVD-ProBD-Clone 3.00 (amd64-unknown-freebsd10.0) Copyright (C) 1995-2010 Jörg
Schilling
Using libscg version 'schily-0.9'
scsibus0:
        0,0,0     0) 'SEAGATE ' 'ST39236LW       ' '0004' Disk
        0,1,0     1) 'SEAGATE ' 'ST39173W        ' '5958' Disk
        0,2,0     2) *
        0,3,0     3) 'iomega  ' 'jaz 1GB         ' 'J.86' Removable Disk
        0,4,0     4) 'NEC     ' 'CD-ROM DRIVE:466' '1.26' Removable CD-ROM
        0,5,0     5) *
        0,6,0     6) *
        0,7,0     7) *
scsibus1:
        1,0,0   100) *
        1,1,0   101) *
        1,2,0   102) *
        1,3,0   103) *
        1,4,0   104) *
        1,5,0   105) 'YAMAHA  ' 'CRW4260         ' '1.0q' Removable CD-ROM
        1,6,0   106) 'ARTEC   ' 'AM12S           ' '1.06' Scanner
        1,7,0   107) *
```

Locate the entry for the CD burner and use the three numbers separated by commas as the value for `dev`. In this case, the Yamaha burner device is `1,5,0`, so the appropriate input to specify that device is `dev=1,5,0`. Refer to the manual page for `cdrecord` for other ways to specify this value and for information on writing audio tracks and controlling the write speed.

Alternately, run the following command to get the device address of the burner:

```
# camcontrol devlist
<MATSHITA CDRW/DVD UJDA740 1.00>   at scbus1 target 0 lun 0 (cd0,pass0)
```

Use the numeric values for `scbus`, `target`, and `lun`. For this example, `1,0,0` is the device name to use.

21.5.3. Writing Data to an ISO File System

In order to produce a data CD, the data files that are going to make up the tracks on the CD must be prepared before they can be burned to the CD. In FreeBSD, sysutils/cdrtools installs `mkisofs`, which can be used to produce an ISO 9660 file system that is an image of a directory tree within a UNIX® file system. The simplest usage is to specify the name of the ISO file to create and the path to the files to place into the ISO 9660 file system:

```
# mkisofs -o imagefile.iso /path/to/tree
```

This command maps the file names in the specified path to names that fit the limitations of the standard ISO 9660 file system, and will exclude files that do not meet the standard for ISO file systems.

A number of options are available to overcome the restrictions imposed by the standard. In particular, `-R` enables the Rock Ridge extensions common to UNIX® systems and `-J` enables Joliet extensions used by Microsoft® systems.

For CDs that are going to be used only on FreeBSD systems, `-U` can be used to disable all filename restrictions. When used with `-R`, it produces a file system image that is identical to the specified FreeBSD tree, even if it violates the ISO 9660 standard.

The last option of general use is `-b`. This is used to specify the location of a boot image for use in producing an "El Torito" bootable CD. This option takes an argument which is the path to a boot image from the top of the tree being written to the CD. By default, `mkisofs` creates an ISO image in "floppy disk emulation" mode, and thus expects the boot image to be exactly 1200, 1440 or 2880 KB in size. Some boot loaders, like the one used by the FreeBSD distribution media, do not use emulation mode. In this case, `-no-emul-boot` should be used. So, if /tmp/myboot holds a bootable FreeBSD system with the boot image in /tmp/myboot/boot/cdboot, this command would produce /tmp/bootable.iso:

```
# mkisofs -R -no-emul-boot -b boot/cdboot -o /tmp/bootable.iso /tmp/myboot
```

The resulting ISO image can be mounted as a memory disk with:

```
# mdconfig -a -t vnode -f /tmp/bootable.iso -u 0
# mount -t cd9660 /dev/md0 /mnt
```

One can then verify that /mnt and /tmp/myboot are identical.

There are many other options available for `mkisofs` to fine-tune its behavior. Refer to mkisofs(8) for details.

> It is possible to copy a data CD to an image file that is functionally equivalent to the image file created with `mkisofs`. To do so, use dd with the device name as the input file and the name of the ISO to create as the output file:
>
> ```
> # dd if=/dev/cd0 of=file.iso bs=2048
> ```
>
> The resulting image file can be burned to CD as described in Burning a CD.

21.5.4. Using Data CDs

Once an ISO has been burned to a CD, it can be mounted by specifying the file system type, the name of the device containing the CD, and an existing mount point:

```
# mount -t cd9660 /dev/cd0 /mnt
```

Since `mount` assumes that a file system is of type `ufs`, an `Incorrect super block` error will occur if `-t cd9660` is not included when mounting a data CD.

While any data CD can be mounted this way, disks with certain ISO 9660 extensions might behave oddly. For example, Joliet disks store all filenames in two-byte Unicode characters. If some non-English characters show up as question marks, specify the local charset with `-C`. For more information, refer to mount_cd9660(8).

> In order to do this character conversion with the help of `-C`, the kernel requires the cd9660_iconv.ko module to be loaded. This can be done either by adding this line to loader.conf:
>
> ```
> cd9660_iconv_load="YES"
> ```
>
> and then rebooting the machine, or by directly loading the module with `kldload`.

Occasionally, `Device not configured` will be displayed when trying to mount a data CD. This usually means that the CD drive has not detected a disk in the tray, or that the drive is not visible on the bus. It can take a couple of seconds for a CD drive to detect media, so be patient.

Sometimes, a SCSICD drive may be missed because it did not have enough time to answer the bus reset. To resolve this, a custom kernel can be created which increases the default SCSI delay. Add the following option to the custom kernel configuration file and rebuild the kernel using the instructions in "Building and Installing a Custom Kernel":

```
options SCSI_DELAY=15000
```

This tells the SCSI bus to pause 15 seconds during boot, to give the CD drive every possible chance to answer the bus reset.

> It is possible to burn a file directly to CD, without creating an ISO 9660 file system. This is known as burning a raw data CD and some people do this for backup purposes.
>
> This type of disk can not be mounted as a normal data CD. In order to retrieve the data burned to such a CD, the data must be read from the raw device node. For example, this command will extract a compressed tar file located on the second CD device into the current working directory:
>
> ```
> # tar xzvf /dev/cd1
> ```
>
> In order to mount a data CD, the data must be written using `mkisofs`.

21.5.5. Duplicating Audio CDs

To duplicate an audio CD, extract the audio data from the CD to a series of files, then write these files to a blank CD.

Duplicating an Audio CD describes how to duplicate and burn an audio CD. If the FreeBSD version is less than 10.0 and the device is ATAPI, the `atapicam` module must be first loaded using the instructions in Supported Devices.

Procedure: Duplicating an Audio CD

1. The sysutils/cdrtools package or port installs `cdda2wav`. This command can be used to extract all of the audio tracks, with each track written to a separate WAV file in the current working directory:

   ```
   % cdda2wav -vall -B -Owav
   ```

   A device name does not need to be specified if there is only one CD device on the system. Refer to the `cdda2wav` manual page for instructions on how to specify a device and to learn more about the other options available for this command.

2. Use `cdrecord` to write the .wav files:

```
% cdrecord -v dev=2,0 -dao -useinfo  *.wav
```

Make sure that *2,0* is set appropriately, as described in Burning a CD.

21.6. Creating and Using DVD Media

Compared to the CD, the DVD is the next generation of optical media storage technology. The DVD can hold more data than any CD and is the standard for video publishing.

Five physical recordable formats can be defined for a recordable DVD:

- DVD-R: This was the first DVD recordable format available. The DVD-R standard is defined by the DVD Forum. This format is write once.
- DVD-RW: This is the rewritable version of the DVD-R standard. A DVD-RW can be rewritten about 1000 times.
- DVD-RAM: This is a rewritable format which can be seen as a removable hard drive. However, this media is not compatible with most DVD-ROM drives and DVD-Video players as only a few DVD writers support the DVD-RAM format. Refer to Using a DVD-RAM for more information on DVD-RAM use.
- DVD+RW: This is a rewritable format defined by the DVD+RW Alliance. A DVD+RW can be rewritten about 1000 times.
- DVD+R: This format is the write once variation of the DVD+RW format.

A single layer recordable DVD can hold up to 4,700,000,000 bytes which is actually 4.38 GB or 4485 MB as 1 kilobyte is 1024 bytes.

A distinction must be made between the physical media and the application. For example, a DVD-Video is a specific file layout that can be written on any recordable DVD physical media such as DVD-R, DVD+R, or DVD-RW. Before choosing the type of media, ensure that both the burner and the DVD-Video player are compatible with the media under consideration.

21.6.1. Configuration

To perform DVD recording, use growisofs(1). This command is part of the sysutils/dvd+rw-tools utilities which support all DVD media types.

These tools use the SCSI subsystem to access the devices, therefore ATAPI/CAM support must be loaded or statically compiled into the kernel. This support is not needed if the burner uses the USB interface. Refer to USB Storage Devices for more details on USB device configuration.

DMA access must also be enabled for ATAPI devices, by adding the following line to /boot/loader.conf:

```
hw.ata.atapi_dma="1"
```

Before attempting to use dvd+rw-tools, consult the Hardware Compatibility Notes.

For a graphical user interface, consider using sysutils/k3b which provides a user friendly interface to growisofs(1) and many other burning tools.

21.6.2. Burning Data DVDs

Since growisofs(1) is a front-end to mkisofs, it will invoke mkisofs(8) to create the file system layout and perform the write on the DVD. This means that an image of the data does not need to be created before the burning process.

To burn to a DVD+R or a DVD-R the data in /path/to/data, use the following command:

```
# growisofs -dvd-compat -Z /dev/cd0 -J -R /path/to/data
```

In this example, `-J -R` is passed to mkisofs(8) to create an ISO 9660 file system with Joliet and Rock Ridge extensions. Refer to mkisofs(8) for more details.

For the initial session recording, `-Z` is used for both single and multiple sessions. Replace */dev/cd0*, with the name of the DVD device. Using `-dvd-compat` indicates that the disk will be closed and that the recording will be unappendable. This should also provide better media compatibility with DVD-ROM drives.

To burn a pre-mastered image, such as *imagefile.iso*, use:

```
# growisofs -dvd-compat -Z /dev/cd0=imagefile.iso
```

The write speed should be detected and automatically set according to the media and the drive being used. To force the write speed, use `-speed=`. Refer to growisofs(1) for example usage.

In order to support working files larger than 4.38GB, an UDF/ISO-9660 hybrid file system must be created by passing `-udf -iso-level 3` to mkisofs(8) and all related programs, such as growisofs(1). This is required only when creating an ISO image file or when writing files directly to a disk. Since a disk created this way must be mounted as an UDF file system with mount_udf(8), it will be usable only on an UDF aware operating system. Otherwise it will look as if it contains corrupted files.

To create this type of ISO file:

```
% mkisofs -R -J -udf -iso-level 3 -o imagefile.iso /path/to/data
```

To burn files directly to a disk:

> ```
> # growisofs -dvd-compat -udf -iso-level 3 -Z /dev/cd0 -J -R
> /path/to/data
> ```
>
> When an ISO image already contains large files, no additional options are required for growisofs(1) to burn that image on a disk.
>
> Be sure to use an up-to-date version of sysutils/cdrtools, which contains mkisofs(8), as an older version may not contain large files support. If the latest version does not work, install sysutils/cdrtools-devel and read its mkisofs(8).

21.6.3. Burning a DVD-Video

A DVD-Video is a specific file layout based on the ISO 9660 and micro-UDF (M-UDF) specifications. Since DVD-Video presents a specific data structure hierarchy, a particular program such as multimedia/dvdauthor is needed to author the DVD.

If an image of the DVD-Video file system already exists, it can be burned in the same way as any other image. If `dvdauthor` was used to make the DVD and the result is in /path/to/video, the following command should be used to burn the DVD-Video:

```
# growisofs -Z /dev/cd0 -dvd-video /path/to/video
```

`-dvd-video` is passed to mkisofs(8) to instruct it to create a DVD-Video file system layout. This option implies the `-dvd-compat` growisofs(1) option.

21.6.4. Using a DVD+RW

Unlike CD-RW, a virgin DVD+RW needs to be formatted before first use. It is *recommended* to let growisofs(1) take care of this automatically whenever appropriate. However, it is possible to use `dvd+rw-format` to format the DVD+RW:

```
# dvd+rw-format /dev/cd0
```

Only perform this operation once and keep in mind that only virgin DVD+RW medias need to be formatted. Once formatted, the DVD+RW can be burned as usual.

To burn a totally new file system and not just append some data onto a DVD+RW, the media does not need to be blanked first. Instead, write over the previous recording like this:

```
# growisofs -Z /dev/cd0 -J -R /path/to/newdata
```

The DVD+RW format supports appending data to a previous recording. This operation consists of merging a new session to the existing one as it is not considered to be multi-session writing. growisofs(1) will *grow* the ISO 9660 file system present on the media.

For example, to append data to a DVD+RW, use the following:

```
# growisofs -M /dev/cd0 -J -R /path/to/nextdata
```

The same mkisofs(8) options used to burn the initial session should be used during next writes.

Use `-dvd-compat` for better media compatibility with DVD-ROM drives. When using DVD+RW, this option will not prevent the addition of data.

To blank the media, use:

```
# growisofs -Z /dev/cd0=/dev/zero
```

21.6.5. Using a DVD-RW

A DVD-RW accepts two disc formats: incremental sequential and restricted overwrite. By default, DVD-RW discs are in sequential format.

A virgin DVD-RW can be directly written without being formatted. However, a non-virgin DVD-RW in sequential format needs to be blanked before writing a new initial session.

To blank a DVD-RW in sequential mode:

```
# dvd+rw-format -blank=full /dev/cd0
```

A full blanking using `-blank=full` will take about one hour on a 1x media. A fast blanking can be performed using `-blank`, if the DVD-RW will be recorded in Disk-At-Once (DAO) mode. To burn the DVD-RW in DAO mode, use the command:

```
# growisofs -use-the-force-luke=dao -Z /dev/cd0=imagefile.iso
```

Since growisofs(1) automatically attempts to detect fast blanked media and engage DAO write, `-use-the-force-luke=dao` should not be required.

One should instead use restricted overwrite mode with any DVD-RW as this format is more flexible than the default of incremental sequential.

To write data on a sequential DVD-RW, use the same instructions as for the other DVD formats:

```
# growisofs -Z /dev/cd0 -J -R /path/to/data
```

To append some data to a previous recording, use `-M` with growisofs(1). However, if data is appended on a DVD-RW in incremental sequential mode, a new session will be created on the disc and the result will be a multi-session disc.

A DVD-RW in restricted overwrite format does not need to be blanked before a new initial session. Instead, overwrite the disc with `-Z`. It is also possible to grow an existing ISO 9660 file system written on the disc with `-M`. The result will be a one-session DVD.

To put a DVD-RW in restricted overwrite format, the following command must be used:

```
# dvd+rw-format /dev/cd0
```

To change back to sequential format, use:

```
# dvd+rw-format -blank=full /dev/cd0
```

21.6.6. Multi-Session

Few DVD-ROM drives support multi-session DVDs and most of the time only read the first session. DVD+R, DVD-R and DVD-RW in sequential format can accept multiple sessions. The notion of multiple sessions does not exist for the DVD+RW and the DVD-RW restricted overwrite formats.

Using the following command after an initial non-closed session on a DVD+R, DVD-R, or DVD-RW in sequential format, will add a new session to the disc:

```
# growisofs -M /dev/cd0 -J -R /path/to/nextdata
```

Using this command with a DVD+RW or a DVD-RW in restricted overwrite mode will append data while merging the new session to the existing one. The result will be a single-session disc. Use this method to add data after an initial write on these types of media.

Since some space on the media is used between each session to mark the end and start of sessions, one should add sessions with a large amount of data to optimize media space. The number of sessions is limited to 154 for a DVD+R, about 2000 for a DVD-R, and 127 for a DVD+R Double Layer.

21.6.7. For More Information

To obtain more information about a DVD, use `dvd+rw-mediainfo /dev/cd0` while the disc in the specified drive.

More information about dvd+rw-tools can be found in growisofs(1), on the dvd+rw-tools web site, and in the cdwrite mailing list archives.

When creating a problem report related to the use of dvd+rw-tools, always include the output of `dvd+rw-mediainfo`.

21.6.8. Using a DVD-RAM

DVD-RAM writers can use either a SCSI or ATAPI interface. For ATAPI devices, DMA access has to be enabled by adding the following line to /boot/loader.conf:

```
hw.ata.atapi_dma="1"
```

A DVD-RAM can be seen as a removable hard drive. Like any other hard drive, the DVD-RAM must be formatted before it can be used. In this example, the whole disk space will be formatted with a standard UFS2 file system:

```
# dd if=/dev/zero of=/dev/acd0 bs=2k count=1
# bsdlabel -Bw acd0
# newfs /dev/acd0
```

The DVD device, acd0, must be changed according to the configuration.

Once the DVD-RAM has been formatted, it can be mounted as a normal hard drive:

```
# mount /dev/acd0 /mnt
```

Once mounted, the DVD-RAM will be both readable and writeable.

21.7. Creating and Using Floppy Disks

This section explains how to format a 3.5 inch floppy disk in FreeBSD.

Procedure: Steps to Format a Floppy

A floppy disk needs to be low-level formatted before it can be used. This is usually done by the vendor, but formatting is a good way to check media integrity. To low-level format the floppy disk on FreeBSD, use fdformat(8). When using this utility, make note of any error messages, as these can help determine if the disk is good or bad.

1. To format the floppy, insert a new 3.5 inch floppy disk into the first floppy drive and issue:

   ```
   # /usr/sbin/fdformat -f 1440 /dev/fd0
   ```

2. After low-level formatting the disk, create a disk label as it is needed by the system to determine the size of the disk and its geometry. The supported geometry values are listed in /etc/disktab.

   To write the disk label, use bsdlabel(8):

```
# /sbin/bsdlabel -B -w /dev/fd0 fd1440
```

3. The floppy is now ready to be high-level formatted with a file system. The floppy's file system can be either UFS or FAT, where FAT is generally a better choice for floppies.

   To format the floppy with FAT, issue:

   ```
   # /sbin/newfs_msdos /dev/fd0
   ```

The disk is now ready for use. To use the floppy, mount it with mount_msdosfs(8). One can also install and use emulators/mtools from the Ports Collection.

21.8. Backup Basics

Implementing a backup plan is essential in order to have the ability to recover from disk failure, accidental file deletion, random file corruption, or complete machine destruction, including destruction of on-site backups.

The backup type and schedule will vary, depending upon the importance of the data, the granularity needed for file restores, and the amount of acceptable downtime. Some possible backup techniques include:

- Archives of the whole system, backed up onto permanent, off-site media. This provides protection against all of the problems listed above, but is slow and inconvenient to restore from, especially for non-privileged users.
- File system snapshots, which are useful for restoring deleted files or previous versions of files.
- Copies of whole file systems or disks which are synchronized with another system on the network using a scheduled net/rsync.
- Hardware or software RAID, which minimizes or avoids downtime when a disk fails.

Typically, a mix of backup techniques is used. For example, one could create a schedule to automate a weekly, full system backup that is stored off-site and to supplement this backup with hourly ZFS snapshots. In addition, one could make a manual backup of individual directories or files before making file edits or deletions.

This section describes some of the utilities which can be used to create and manage backups on a FreeBSD system.

21.8.1. File System Backups

The traditional UNIX® programs for backing up a file system are dump(8), which creates the backup, and restore(8), which restores the backup. These utilities work at the disk block level, below the abstractions of the files, links, and directories that are created by file systems. Unlike other backup software, `dump` backs up an entire file system and is unable to backup only part of a file system or a directory tree that spans multiple file systems. Instead of writing files and directories, `dump` writes the raw data blocks that comprise files and directories.

> If `dump` is used on the root directory, it will not back up /home, /usr, or many other directories since these are typically mount points for other file systems or symbolic links into those file systems.

When used to restore data, `restore` stores temporary files in /tmp/ by default. When using a recovery disk with a small /tmp, set `TMPDIR` to a directory with more free space for the restore to succeed.

When using `dump`, be aware that some quirks remain from its early days in Version 6 of AT&T UNIX®,circa 1975. The default parameters assume a backup to a 9-track tape, rather than to another type of media or to the high-density tapes available today. These defaults must be overridden on the command line.

It is possible to backup a file system across the network to another system or a tape drive attached to another computer. While the rdump(8) and rrestore(8) utilities can be used for this purpose, they are not considered to be secure.

Instead, one can use `dump` and `restore` more securely over an SSH connection. This example creates a full, compressed backup of /usr and sends the backup file to the specified host over an SSH connection.

Example 26. Using `dump` over ssh

```
# /sbin/dump -0uan -f - /usr | gzip -2 | ssh -c blowfish \
          targetuser@targetmachine.example.com dd of=/mybigfiles/dump-usr-l0.gz
```

This example sets `RSH` in order to write the backup to a tape drive on a remote system over an SSH connection:

Example 27. Using `dump` over ssh with `RSH` Set

```
# env RSH=/usr/bin/ssh /sbin/dump -0uan -f
targetuser@targetmachine.example.com:/dev/sa0 /usr
```

> Systems using the Z file system (ZFS) can make use of zfs(8) for creating snapshots, as well as sending and receiving them to/from remote systems.

21.8.2. Directory Backups

Several built-in utilities are available for backing up and restoring specified files and directories as needed.

A good choice for making a backup of all of the files in a directory is tar(1). This utility dates back to

Version 6 of AT&T UNIX® and by default assumes a recursive backup to a local tape device. Switches can be used to instead specify the name of a backup file.

This example creates a compressed backup of the current directory and saves it to /tmp/mybackup.tgz. When creating a backup file, make sure that the backup is not saved to the same directory that is being backed up.

Example 28. Backing Up the Current Directory with `tar`

```
# tar czvf /tmp/mybackup.tgz .
```

To restore the entire backup, `cd` into the directory to restore into and specify the name of the backup. Note that this will overwrite any newer versions of files in the restore directory. When in doubt, restore to a temporary directory or specify the name of the file within the backup to restore.

Example 29. Restoring Up the Current Directory with `tar`

```
# tar xzvf /tmp/mybackup.tgz
```

There are dozens of available switches which are described in tar(1). This utility also supports the use of exclude patterns to specify which files should not be included when backing up the specified directory or restoring files from a backup.

To create a backup using a specified list of files and directories, cpio(1) is a good choice. Unlike `tar`, `cpio` does not know how to walk the directory tree and it must be provided the list of files to backup.

For example, a list of files can be created using `ls` or `find`. This example creates a recursive listing of the current directory which is then piped to `cpio` in order to create an output backup file named /tmp/mybackup.cpio.

Example 30. Using `ls` and `cpio` to Make a Recursive Backup of the Current Directory

```
# ls -R | cpio -ovF /tmp/mybackup.cpio
```

A backup utility which tries to bridge the features provided by `tar` and `cpio` is pax(1). Over the years, the various versions of `tar` and `cpio` became slightly incompatible. POSIX® created `pax` which attempts to read and write many of the various `cpio` and `tar` formats, plus new formats of its own.

The `pax` equivalent to the previous examples would be:

Example 31. Backing Up the Current Directory with `pax`

```
# pax -wf /tmp/mybackup.pax .
```

21.8.3. Using Data Tapes for Backups

While tape technology has continued to evolve, modern backup systems tend to combine off-site backups with local removable media. FreeBSD supports any tape drive that uses SCSI, such as LTO or DAT. There is limited support for SATA and USB tape drives.

For SCSI tape devices, FreeBSD uses the sa(4) driver and the /dev/sa0, /dev/nsa0, and /dev/esa0 devices. The physical device name is /dev/sa0. When /dev/nsa0 is used, the backup application will not rewind the tape after writing a file, which allows writing more than one file to a tape. Using /dev/esa0 ejects the tape after the device is closed.

In FreeBSD, `mt` is used to control operations of the tape drive, such as seeking through files on a tape or writing tape control marks to the tape. For example, the first three files on a tape can be preserved by skipping past them before writing a new file:

```
# mt -f /dev/nsa0 fsf 3
```

This utility supports many operations. Refer to mt(1) for details.

To write a single file to tape using `tar`, specify the name of the tape device and the file to backup:

```
# tar cvf /dev/sa0 file
```

To recover files from a `tar` archive on tape into the current directory:

```
# tar xvf /dev/sa0
```

To backup a UFS file system, use `dump`. This examples backs up /usr without rewinding the tape when finished:

```
# dump -0aL -b64 -f /dev/nsa0 /usr
```

To interactively restore files from a `dump` file on tape into the current directory:

```
# restore -i -f /dev/nsa0
```

21.8.4. Third-Party Backup Utilities

The FreeBSD Ports Collection provides many third-party utilities which can be used to schedule the creation of backups, simplify tape backup, and make backups easier and more convenient. Many of these applications are client/server based and can be used to automate the backups of a single system or all of the computers in a network.

Popular utilities include:

- Amanda (misc/amanda-server and misc/amanda-client),
- Bacula (sysutils/bacula13-server and sysutils/bacula13-client),
- Bareos (sysutils/bareos-server and sysutils/bareos-client),
- net/rsync,
- sysutils/duply, and
- sysutils/duplicity.

21.8.5. Emergency Recovery

In addition to regular backups, it is recommended to perform the following steps as part of an emergency preparedness plan.

Create a print copy of the output of the following commands:

- `gpart show`
- `more /etc/fstab`
- `pkg prime-list`
- `dmesg`

Store this printout and a copy of the installation media in a secure location. Should an emergency restore be needed, boot into the installation media and select `Live CD` to access a rescue shell. This rescue mode can be used to view the current state of the system, and if needed, to reformat disks and restore data from backups.

Next, test the rescue shell and the backups. Make notes of the procedure. Store these notes with the media, the printouts, and the backups. These notes may prevent the inadvertent destruction of the backups while under the stress of performing an emergency recovery.

For an added measure of security, store the latest backup at a remote location which is physically separated from the computers and disk drives by a significant distance.

21.9. Memory Disks

In addition to physical disks, FreeBSD also supports the creation and use of memory disks. One possible use for a memory disk is to access the contents of an ISO file system without the overhead of first burning it to a CD or DVD, then mounting the CD/DVD media.

In FreeBSD, the md(4) driver is used to provide support for memory disks. The GENERIC kernel includes this driver. When using a custom kernel configuration file, ensure it includes this line:

```
device md
```

21.9.1. Attaching and Detaching Existing Images

To mount an existing file system image, use `mdconfig` to specify the name of the ISO file and a free unit number. Then, refer to that unit number to mount it on an existing mount point. Once mounted, the files in the ISO will appear in the mount point. This example attaches *diskimage.iso* to the memory device /dev/md0 then mounts that memory device on /mnt:

```
# mdconfig -f diskimage.iso -u 0
# mount -t cd9660 /dev/md0 /mnt
```

Notice that `-t cd9660` was used to mount an ISO format. If a unit number is not specified with `-u`, `mdconfig` will automatically allocate an unused memory device and output the name of the allocated unit, such as md4. Refer to mdconfig(8) for more details about this command and its options.

When a memory disk is no longer in use, its resources should be released back to the system. First, unmount the file system, then use `mdconfig` to detach the disk from the system and release its resources. To continue this example:

```
# umount /mnt
# mdconfig -d -u 0
```

To determine if any memory disks are still attached to the system, type `mdconfig -l`.

21.9.2. Creating a File- or Memory-Backed Memory Disk

FreeBSD also supports memory disks where the storage to use is allocated from either a hard disk or an area of memory. The first method is commonly referred to as a file-backed file system and the second method as a memory-backed file system. Both types can be created using `mdconfig`.

To create a new memory-backed file system, specify a type of `swap` and the size of the memory disk to create. Then, format the memory disk with a file system and mount as usual. This example creates a 5M memory disk on unit `1`. That memory disk is then formatted with the UFS file system before it is mounted:

```
# mdconfig -a -t swap -s 5m -u 1
# newfs -U md1
/dev/md1: 5.0MB (10240 sectors) block size 16384, fragment size 2048
        using 4 cylinder groups of 1.27MB, 81 blks, 192 inodes.
        with soft updates
super-block backups (for fsck -b #) at:
 160, 2752, 5344, 7936
```

```
# mount /dev/md1 /mnt
# df /mnt
Filesystem 1K-blocks Used Avail Capacity  Mounted on
/dev/md1        4718    4  4338     0%    /mnt
```

To create a new file-backed memory disk, first allocate an area of disk to use. This example creates an empty 5MB file named newimage:

```
# dd if=/dev/zero of=newimage bs=1k count=5k
5120+0 records in
5120+0 records out
```

Next, attach that file to a memory disk, label the memory disk and format it with the UFS file system, mount the memory disk, and verify the size of the file-backed disk:

```
# mdconfig -f newimage -u 0
# bsdlabel -w md0 auto
# newfs -U md0a
/dev/md0a: 5.0MB (10224 sectors) block size 16384, fragment size 2048
        using 4 cylinder groups of 1.25MB, 80 blks, 192 inodes.
super-block backups (for fsck -b #) at:
 160, 2720, 5280, 7840
# mount /dev/md0a /mnt
# df /mnt
Filesystem 1K-blocks Used Avail Capacity  Mounted on
/dev/md0a       4710    4  4330     0%    /mnt
```

It takes several commands to create a file- or memory-backed file system using `mdconfig`. FreeBSD also comes with `mdmfs` which automatically configures a memory disk, formats it with the UFS file system, and mounts it. For example, after creating *newimage* with `dd`, this one command is equivalent to running the `bsdlabel`, `newfs`, and `mount` commands shown above:

```
# mdmfs -F newimage -s 5m md0 /mnt
```

To instead create a new memory-based memory disk with `mdmfs`, use this one command:

```
# mdmfs -s 5m md1 /mnt
```

If the unit number is not specified, `mdmfs` will automatically select an unused memory device. For more details about `mdmfs`, refer to mdmfs(8).

21.10. File System Snapshots

FreeBSD offers a feature in conjunction with soft updates: file system snapshots.

UFS snapshots allow a user to create images of specified file systems, and treat them as a file. When using the Z file system (ZFS), refer to Managing Snapshots on how to use snapshots.

Snapshot files must be created in the file system that the action is performed on, and a user may create no more than 20 snapshots per file system. Active snapshots are recorded in the superblock so they are persistent across unmount and remount operations along with system reboots. When a snapshot is no longer required, it can be removed using rm(1). While snapshots may be removed in any order, all the used space may not be acquired because another snapshot will possibly claim some of the released blocks.

The un-alterable `snapshot` file flag is set by mksnap_ffs(8) after initial creation of a snapshot file. unlink(1) makes an exception for snapshot files since it allows them to be removed.

Snapshots are created using mount(8). To place a snapshot of /var in the file /var/snapshot/snap, use the following command:

```
# mount -u -o snapshot /var/snapshot/snap /var
```

Alternatively, use mksnap_ffs(8) to create the snapshot:

```
# mksnap_ffs /var /var/snapshot/snap
```

One can find snapshot files on a file system, such as /var, using find(1):

```
# find /var -flags snapshot
```

Once a snapshot has been created, it has several uses:

- Some administrators will use a snapshot file for backup purposes, because the snapshot can be transferred to CDs or tape.
- The file system integrity checker, fsck(8), may be run on the snapshot. Assuming that the file system was clean when it was mounted, this should always provide a clean and unchanging result.
- Running dump(8) on the snapshot will produce a dump file that is consistent with the file system and the timestamp of the snapshot. dump(8) can also take a snapshot, create a dump image, and then remove the snapshot in one command by using `-L`.
- The snapshot can be mounted as a frozen image of the file system. To mount(8) the snapshot /var/snapshot/snap run:

  ```
  # mdconfig -a -t vnode -o readonly -f /var/snapshot/snap -u 4
  # mount -r /dev/md4 /mnt
  ```

The frozen /var is now available through /mnt. Everything will initially be in the same state it was during the snapshot creation time. The only exception is that any earlier snapshots will appear as zero length files. To unmount the snapshot, use:

```
# umount /mnt
# mdconfig -d -u 4
```

For more information about `softupdates` and file system snapshots, including technical papers, visit Marshall Kirk McKusick's website at http://www.mckusick.com/.

21.11. Disk Quotas

Disk quotas can be used to limit the amount of disk space or the number of files a user or members of a group may allocate on a per-file system basis. This prevents one user or group of users from consuming all of the available disk space.

This section describes how to configure disk quotas for the UFS file system. To configure quotas on the ZFS file system, refer to Dataset, User, and Group Quotas

21.11.1. Enabling Disk Quotas

To determine if the FreeBSD kernel provides support for disk quotas:

```
% sysctl kern.features.ufs_quota
kern.features.ufs_quota: 1
```

In this example, the `1` indicates quota support. If the value is instead `0`, add the following line to a custom kernel configuration file and rebuild the kernel using the instructions in Configuring the FreeBSD Kernel:

```
options QUOTA
```

Next, enable disk quotas in /etc/rc.conf:

```
quota_enable="YES"
```

Normally on bootup, the quota integrity of each file system is checked by quotacheck(8). This program insures that the data in the quota database properly reflects the data on the file system. This is a time consuming process that will significantly affect the time the system takes to boot. To skip this step, add this variable to /etc/rc.conf:

```
check_quotas="NO"
```

Finally, edit /etc/fstab to enable disk quotas on a per-file system basis. To enable per-user quotas on a file system, add `userquota` to the options field in the /etc/fstab entry for the file system to enable quotas on. For example:

```
/dev/da1s2g   /home    ufs rw,userquota 1 2
```

To enable group quotas, use `groupquota` instead. To enable both user and group quotas, separate the options with a comma:

```
/dev/da1s2g    /home    ufs rw,userquota,groupquota 1 2
```

By default, quota files are stored in the root directory of the file system as quota.user and quota.group. Refer to fstab(5) for more information. Specifying an alternate location for the quota files is not recommended.

Once the configuration is complete, reboot the system and /etc/rc will automatically run the appropriate commands to create the initial quota files for all of the quotas enabled in /etc/fstab.

In the normal course of operations, there should be no need to manually run quotacheck(8), quotaon(8), or quotaoff(8). However, one should read these manual pages to be familiar with their operation.

21.11.2. Setting Quota Limits

To verify that quotas are enabled, run:

```
# quota -v
```

There should be a one line summary of disk usage and current quota limits for each file system that quotas are enabled on.

The system is now ready to be assigned quota limits with `edquota`.

Several options are available to enforce limits on the amount of disk space a user or group may allocate, and how many files they may create. Allocations can be limited based on disk space (block quotas), number of files (inode quotas), or a combination of both. Each limit is further broken down into two categories: hard and soft limits.

A hard limit may not be exceeded. Once a user reaches a hard limit, no further allocations can be made on that file system by that user. For example, if the user has a hard limit of 500 kbytes on a file system and is currently using 490 kbytes, the user can only allocate an additional 10 kbytes. Attempting to allocate an additional 11 kbytes will fail.

Soft limits can be exceeded for a limited amount of time, known as the grace period, which is one week by default. If a user stays over their limit longer than the grace period, the soft limit turns into a hard limit and no further allocations are allowed. When the user drops back below the soft limit, the grace period is reset.

In the following example, the quota for the `test` account is being edited. When `edquota` is invoked, the editor specified by `EDITOR` is opened in order to edit the quota limits. The default editor is set to vi.

```
# edquota -u test
Quotas for user test:
/usr: kbytes in use: 65, limits (soft = 50, hard = 75)
        inodes in use: 7, limits (soft = 50, hard = 60)
/usr/var: kbytes in use: 0, limits (soft = 50, hard = 75)
        inodes in use: 0, limits (soft = 50, hard = 60)
```

There are normally two lines for each file system that has quotas enabled. One line represents the block limits and the other represents the inode limits. Change the value to modify the quota limit. For example, to raise the block limit on /usr to a soft limit of `500` and a hard limit of `600`, change the values in that line as follows:

```
/usr: kbytes in use: 65, limits (soft = 500, hard = 600)
```

The new quota limits take effect upon exiting the editor.

Sometimes it is desirable to set quota limits on a range of users. This can be done by first assigning the desired quota limit to a user. Then, use `-p` to duplicate that quota to a specified range of user IDs (UIDs). The following command will duplicate those quota limits for UIDs `10,000` through `19,999`:

```
# edquota -p test 10000-19999
```

For more information, refer to edquota(8).

21.11.3. Checking Quota Limits and Disk Usage

To check individual user or group quotas and disk usage, use quota(1). A user may only examine their own quota and the quota of a group they are a member of. Only the superuser may view all user and group quotas. To get a summary of all quotas and disk usage for file systems with quotas enabled, use repquota(8).

Normally, file systems that the user is not using any disk space on will not show in the output of `quota`, even if the user has a quota limit assigned for that file system. Use `-v` to display those file systems. The following is sample output from `quota -v` for a user that has quota limits on two file systems.

```
Disk quotas for user test (uid 1002):
     Filesystem  usage    quota   limit   grace   files   quota   limit   grace
           /usr      65*     50      75   5days       7      50      60
       /usr/var       0      50      75               0      50      60
```

In this example, the user is currently 15 kbytes over the soft limit of 50 kbytes on /usr and has 5 days of grace period left. The asterisk `*` indicates that the user is currently over the quota limit.

21.11.4. Quotas over NFS

Quotas are enforced by the quota subsystem on the NFS server. The rpc.rquotad(8) daemon makes quota information available to `quota` on NFS clients, allowing users on those machines to see their quota statistics.

On the NFS server, enable `rpc.rquotad` by removing the `#` from this line in **/etc/inetd.conf**:

```
rquotad/1      dgram rpc/udp wait root /usr/libexec/rpc.rquotad rpc.rquotad
```

Then, restart `inetd`:

```
# service inetd restart
```

21.12. Encrypting Disk Partitions

FreeBSD offers excellent online protections against unauthorized data access. File permissions and Mandatory Access Control (MAC) help prevent unauthorized users from accessing data while the operating system is active and the computer is powered up. However, the permissions enforced by the operating system are irrelevant if an attacker has physical access to a computer and can move the computer's hard drive to another system to copy and analyze the data.

Regardless of how an attacker may have come into possession of a hard drive or powered-down computer, the GEOM-based cryptographic subsystems built into FreeBSD are able to protect the data on the computer's file systems against even highly-motivated attackers with significant resources. Unlike encryption methods that encrypt individual files, the built-in `gbde` and `geli` utilities can be used to transparently encrypt entire file systems. No cleartext ever touches the hard drive's platter.

This chapter demonstrates how to create an encrypted file system on FreeBSD. It first demonstrates the process using `gbde` and then demonstrates the same example using `geli`.

GBDE disk-encryption is deprecated, and removed in FreeBSD 15.0.

21.12.1. Disk Encryption with gbde

The objective of the gbde(4) facility is to provide a formidable challenge for an attacker to gain access to the contents of a *cold* storage device. However, if the computer is compromised while up and running and the storage device is actively attached, or the attacker has access to a valid passphrase, it offers no protection to the contents of the storage device. Thus, it is important to provide physical security while the system is running and to protect the passphrase used by the encryption mechanism.

This facility provides several barriers to protect the data stored in each disk sector. It encrypts the contents of a disk sector using 128-bit AES in CBC mode. Each sector on the disk is encrypted with a different AES key. For more information on the cryptographic design, including how the sector keys are derived from the user-supplied passphrase, refer to gbde(4).

FreeBSD provides a kernel module for gbde which can be loaded with this command:

```
# kldload geom_bde
```

If using a custom kernel configuration file, ensure it contains this line:

`options GEOM_BDE`

The following example demonstrates adding a new hard drive to a system that will hold a single encrypted partition that will be mounted as /private.

Procedure: Encrypting a Partition with gbde

1. Add the New Hard Drive

   Install the new drive to the system as explained in Adding Disks. For the purposes of this example, a new hard drive partition has been added as /dev/ad4s1c and /dev/ad0s1* represents the existing standard FreeBSD partitions.

   ```
   # ls /dev/ad*
   /dev/ad0        /dev/ad0s1b     /dev/ad0s1e     /dev/ad4s1
   /dev/ad0s1      /dev/ad0s1c     /dev/ad0s1f     /dev/ad4s1c
   /dev/ad0s1a     /dev/ad0s1d     /dev/ad4
   ```

2. Create a Directory to Hold `gbde` Lock Files

   ```
   # mkdir /etc/gbde
   ```

   The gbde lock file contains information that gbde requires to access encrypted partitions. Without access to the lock file, gbde will not be able to decrypt the data contained in the encrypted partition without significant manual intervention which is not supported by the software. Each encrypted partition uses a separate lock file.

3. Initialize the `gbde` Partition

   A gbde partition must be initialized before it can be used. This initialization needs to be performed only once. This command will open the default editor, in order to set various configuration options in a template. For use with the UFS file system, set the sector_size to 2048:

   ```
   # gbde init /dev/ad4s1c -i -L /etc/gbde/ad4s1c.lock
   #
   # Sector size is the smallest unit of data which can be read or written.
   # Making it too small decreases performance and decreases available space.
   # Making it too large may prevent filesystems from working.  512 is the
   # minimum and always safe.  For UFS, use the fragment size
   #
   ```

```
sector_size =   2048
[...]
```

Once the edit is saved, the user will be asked twice to type the passphrase used to secure the data. The passphrase must be the same both times. The ability of gbde to protect data depends entirely on the quality of the passphrase. For tips on how to select a secure passphrase that is easy to remember, see http://world.std.com/~reinhold/diceware.htm.

This initialization creates a lock file for the gbde partition. In this example, it is stored as /etc/gbde/ad4s1c.lock. Lock files must end in ".lock" in order to be correctly detected by the /etc/rc.d/gbde start up script.

> Lock files *must* be backed up together with the contents of any encrypted partitions. Without the lock file, the legitimate owner will be unable to access the data on the encrypted partition.

4. Attach the Encrypted Partition to the Kernel

   ```
   # gbde attach /dev/ad4s1c -l /etc/gbde/ad4s1c.lock
   ```

   This command will prompt to input the passphrase that was selected during the initialization of the encrypted partition. The new encrypted device will appear in /dev as /dev/device_name.bde:

   ```
   # ls /dev/ad*
   /dev/ad0        /dev/ad0s1b     /dev/ad0s1e     /dev/ad4s1
   /dev/ad0s1      /dev/ad0s1c     /dev/ad0s1f     /dev/ad4s1c
   /dev/ad0s1a     /dev/ad0s1d     /dev/ad4        /dev/ad4s1c.bde
   ```

5. Create a File System on the Encrypted Device

   Once the encrypted device has been attached to the kernel, a file system can be created on the device. This example creates a UFS file system with soft updates enabled. Be sure to specify the partition which has a *.bde extension:

   ```
   # newfs -U /dev/ad4s1c.bde
   ```

6. Mount the Encrypted Partition

   Create a mount point and mount the encrypted file system:

   ```
   # mkdir /private
   # mount /dev/ad4s1c.bde /private
   ```

7. Verify That the Encrypted File System is Available

The encrypted file system should now be visible and available for use:

```
% df -H
Filesystem        Size   Used  Avail Capacity  Mounted on
/dev/ad0s1a      1037M    72M   883M     8%    /
/devfs            1.0K   1.0K     0B   100%    /dev
/dev/ad0s1f       8.1G    55K   7.5G     0%    /home
/dev/ad0s1e      1037M   1.1M   953M     0%    /tmp
/dev/ad0s1d       6.1G   1.9G   3.7G    35%    /usr
/dev/ad4s1c.bde   150G   4.1K   138G     0%    /private
```

After each boot, any encrypted file systems must be manually re-attached to the kernel, checked for errors, and mounted, before the file systems can be used. To configure these steps, add the following lines to /etc/rc.conf:

```
gbde_autoattach_all="YES"
gbde_devices="ad4s1c"
gbde_lockdir="/etc/gbde"
```

This requires that the passphrase be entered at the console at boot time. After typing the correct passphrase, the encrypted partition will be mounted automatically. Additional gbde boot options are available and listed in rc.conf(5).

sysinstall is incompatible with gbde-encrypted devices. All *.bde devices must be detached from the kernel before starting sysinstall or it will crash during its initial probing for devices. To detach the encrypted device used in the example, use the following command:

```
# gbde detach /dev/ad4s1c
```

21.12.2. Disk Encryption with `geli`

An alternative cryptographic GEOM class is available using `geli`. This control utility adds some features and uses a different scheme for doing cryptographic work. It provides the following features:

- Utilizes the crypto(9) framework and automatically uses cryptographic hardware when it is available.
- Supports multiple cryptographic algorithms such as AES-XTS, AES-CBC, and Camellia-CBC.
- Allows the root partition to be encrypted. The passphrase used to access the encrypted root partition will be requested during system boot.
- Allows the use of two independent keys.
- It is fast as it performs simple sector-to-sector encryption.
- Allows backup and restore of master keys. If a user destroys their keys, it is still possible to get

access to the data by restoring keys from the backup.

- Allows a disk to attach with a random, one-time key which is useful for swap partitions and temporary file systems.

More features and usage examples can be found in geli(8).

The following example describes how to generate a key file which will be used as part of the master key for the encrypted provider mounted under /private. The key file will provide some random data used to encrypt the master key. The master key will also be protected by a passphrase. The provider's sector size will be 4kB. The example describes how to attach to the `geli` provider, create a file system on it, mount it, work with it, and finally, how to detach it.

Procedure: Encrypting a Partition with `geli`

1. Load `geli` Support

   Support for `geli` is available as a loadable kernel module. To configure the system to automatically load the module at boot time, add the following line to /boot/loader.conf:

   ```
   geom_eli_load="YES"
   ```

   To load the kernel module now:

   ```
   # kldload geom_eli
   ```

   For a custom kernel, ensure the kernel configuration file contains these lines:

   ```
   options GEOM_ELI
   device crypto
   ```

2. Generate the Master Key

   The following commands generate a master key that all data will be encrypted with. This key can never be changed. Rather than using it directly, it is encrypted with one or more user keys. The user keys are made up of an optional combination of random bytes from a file, /root/da2.key, and/or a passphrase. In this case, the data source for the key file is /dev/random. This command also configures the sector size of the provider (/dev/da2.eli) as 4kB, for better performance:

   ```
   # dd if=/dev/random of=/root/da2.key bs=64 count=1
   # geli init -K /root/da2.key -s 4096 /dev/da2
   Enter new passphrase:
   Reenter new passphrase:
   ```

   It is not mandatory to use both a passphrase and a key file as either method of securing the master key can be used in isolation.

If the key file is given as "-", standard input will be used. For example, this command generates three key files:

```
# cat keyfile1 keyfile2 keyfile3 | geli init -K - /dev/da2
```

3. Attach the Provider with the Generated Key

   To attach the provider, specify the key file, the name of the disk, and the passphrase:

   ```
   # geli attach -k /root/da2.key /dev/da2
   Enter passphrase:
   ```

   This creates a new device with an .eli extension:

   ```
   # ls /dev/da2*
   /dev/da2  /dev/da2.eli
   ```

4. Create the New File System

   Next, format the device with the UFS file system and mount it on an existing mount point:

   ```
   # dd if=/dev/random of=/dev/da2.eli bs=1m
   # newfs /dev/da2.eli
   # mount /dev/da2.eli /private
   ```

   The encrypted file system should now be available for use:

   ```
   # df -H
   Filesystem     Size   Used  Avail Capacity  Mounted on
   /dev/ad0s1a    248M    89M   139M    38%    /
   /devfs         1.0K   1.0K     0B   100%    /dev
   /dev/ad0s1f    7.7G   2.3G   4.9G    32%    /usr
   /dev/ad0s1d    989M   1.5M   909M     0%    /tmp
   /dev/ad0s1e    3.9G   1.3G   2.3G    35%    /var
   /dev/da2.eli   150G   4.1K   138G     0%    /private
   ```

Once the work on the encrypted partition is done, and the /private partition is no longer needed, it is prudent to put the device into cold storage by unmounting and detaching the `geli` encrypted partition from the kernel:

```
# umount /private
# geli detach da2.eli
```

An rc.d script is provided to simplify the mounting of `geli`-encrypted devices at boot time. For this example, add these lines to /etc/rc.conf:

```
geli_devices="da2"
geli_da2_flags="-k /root/da2.key"
```

This configures /dev/da2 as a `geli` provider with a master key of /root/da2.key. The system will automatically detach the provider from the kernel before the system shuts down. During the startup process, the script will prompt for the passphrase before attaching the provider. Other kernel messages might be shown before and after the password prompt. If the boot process seems to stall, look carefully for the password prompt among the other messages. Once the correct passphrase is entered, the provider is attached. The file system is then mounted, typically by an entry in /etc/fstab. Refer to "Mounting and Unmounting File Systems" for instructions on how to configure a file system to mount at boot time.

21.13. Encrypting Swap

Like the encryption of disk partitions, encryption of swap space is used to protect sensitive information. Consider an application that deals with passwords. As long as these passwords stay in physical memory, they are not written to disk and will be cleared after a reboot. However, if FreeBSD starts swapping out memory pages to free space, the passwords may be written to the disk unencrypted. Encrypting swap space can be a solution for this scenario.

This section demonstrates how to configure an encrypted swap partition using gbde(8) or geli(8) encryption. It assumes that /dev/ada0s1b is the swap partition.

21.13.1. Configuring Encrypted Swap

Swap partitions are not encrypted by default and should be cleared of any sensitive data before continuing. To overwrite the current swap partition with random garbage, execute the following command:

```
# dd if=/dev/random of=/dev/ada0s1b bs=1m
```

To encrypt the swap partition using gbde(8), add the `.bde` suffix to the swap line in /etc/fstab:

```
# Device        Mountpoint  FStype  Options     Dump    Pass#
/dev/ada0s1b.bde    none        swap    sw      0   0
```

To instead encrypt the swap partition using geli(8), initialize the device with a onetime key:

```
# geom eli onetime -d /dev/ada0s1b
```

Afterwards, add a line for the device with the `.eli` suffix to /etc/fstab:

```
# Device        Mountpoint  FStype  Options     Dump    Pass#
/dev/ada0s1b.eli    none        swap    sw      0   0
```

By default, geli(8) uses the AES algorithm with a key length of 128 bits. Normally the default settings will suffice. If desired, these defaults can be altered in the options field in /etc/fstab. The possible flags are:

aalgo

Data integrity verification algorithm used to ensure that the encrypted data has not been tampered with. See geli(8) for a list of supported algorithms.

ealgo

Encryption algorithm used to protect the data. See geli(8) for a list of supported algorithms.

keylen

The length of the key used for the encryption algorithm. See geli(8) for the key lengths that are supported by each encryption algorithm.

sectorsize

The size of the blocks data is broken into before it is encrypted. Larger sector sizes increase performance at the cost of higher storage overhead. The recommended size is 4096 bytes.

This example configures an encrypted swap partition using the AES-XTS algorithm with a key length of 128 bits and a sectorsize of 4 kilobytes:

```
# Device        Mountpoint  FStype  Options             Dump    Pass#
/dev/ada0s1b.eli    none        swap    sw,ealgo=AES-XTS,keylen=128,sectorsize=4096 0 0
```

21.13.2. Encrypted Swap Verification

Once the system has rebooted, proper operation of the encrypted swap can be verified using `swapinfo`.

If gbde(8) is being used:

```
% swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ada0s1b.bde   542720        0   542720     0
```

If geli(8) is being used:

```
% swapinfo
Device          1K-blocks     Used    Avail Capacity
/dev/ada0s1b.eli   542720        0   542720     0
```

21.14. Highly Available Storage (HAST)

High availability is one of the main requirements in serious business applications and highly-available storage is a key component in such environments. In FreeBSD, the Highly Available STorage (HAST) framework allows transparent storage of the same data across several physically separated machines connected by a TCP/IP network. HAST can be understood as a network-based RAID1 (mirror), and is similar to the DRBD® storage system used in the GNU/Linux® platform. In combination with other high-availability features of FreeBSD like CARP, HAST makes it possible to build a highly-available storage cluster that is resistant to hardware failures.

The following are the main features of HAST:

- Can be used to mask I/O errors on local hard drives.
- File system agnostic as it works with any file system supported by FreeBSD.
- Efficient and quick resynchronization as only the blocks that were modified during the downtime of a node are synchronized.
- Can be used in an already deployed environment to add additional redundancy.
- Together with CARP, Heartbeat, or other tools, it can be used to build a robust and durable storage system.

Read this chapter to learn:

- What HAST is, how it works, and which features it provides.
- How to set up and use HAST on FreeBSD.
- How to integrate CARP and devd(8) to build a robust storage system.

Before reading this section:

- Understand UNIX® and FreeBSD basics (FreeBSD Basics).
- Know how to configure network interfaces and other core FreeBSD subsystems (Configuration and Tuning).
- Have a good understanding of FreeBSD networking (Network Communication).

The HAST project was sponsored by The FreeBSD Foundation with support from http://www.omc.net/ and http://www.transip.nl/.

21.14.1. HAST Operation

HAST provides synchronous block-level replication between two physical machines: the *primary* node and the *secondary* node. These two machines together are referred to as a cluster.

Since HAST works in a primary-secondary configuration, it allows only one of the cluster nodes to be active at any given time. The primary node, also called *active*, is the one which will handle all the I/O requests to HAST-managed devices. The secondary node is automatically synchronized from the primary node.

The physical components of the HAST system are the local disk on primary node, and the disk on the remote, secondary node.

HAST operates synchronously on a block level, making it transparent to file systems and applications. HAST provides regular GEOM providers in /dev/hast/ for use by other tools or applications. There is no difference between using HAST-provided devices and raw disks or partitions.

Each write, delete, or flush operation is sent to both the local disk and to the remote disk over TCP/IP. Each read operation is served from the local disk, unless the local disk is not up-to-date or an I/O error occurs. In such cases, the read operation is sent to the secondary node.

HAST tries to provide fast failure recovery. For this reason, it is important to reduce synchronization time after a node's outage. To provide fast synchronization, HAST manages an on-disk bitmap of dirty extents and only synchronizes those during a regular synchronization, with an exception of the initial sync.

There are many ways to handle synchronization. HAST implements several replication modes to handle different synchronization methods:

- *memsync*: This mode reports a write operation as completed when the local write operation is finished and when the remote node acknowledges data arrival, but before actually storing the data. The data on the remote node will be stored directly after sending the acknowledgement. This mode is intended to reduce latency, but still provides good reliability. This mode is the default.
- *fullsync*: This mode reports a write operation as completed when both the local write and the remote write complete. This is the safest and the slowest replication mode.
- *async*: This mode reports a write operation as completed when the local write completes. This is the fastest and the most dangerous replication mode. It should only be used when replicating to a distant node where latency is too high for other modes.

21.14.2. HAST Configuration

The HAST framework consists of several components:

- The hastd(8) daemon which provides data synchronization. When this daemon is started, it will automatically load `geom_gate.ko`.
- The userland management utility, hastctl(8).
- The hast.conf(5) configuration file. This file must exist before starting hastd.

Users who prefer to statically build `GEOM_GATE` support into the kernel should add this line to the custom kernel configuration file, then rebuild the kernel using the instructions in Configuring the FreeBSD Kernel:

```
options GEOM_GATE
```

The following example describes how to configure two nodes in primary-secondary operation using HAST to replicate the data between the two. The nodes will be called `hasta`, with an IP address of

`172.16.0.1`, and `hastb`, with an IP address of `172.16.0.2`. Both nodes will have a dedicated hard drive /dev/ad6 of the same size for HAST operation. The HAST pool, sometimes referred to as a resource or the GEOM provider in /dev/hast/, will be called `test`.

Configuration of HAST is done using /etc/hast.conf. This file should be identical on both nodes. The simplest configuration is:

```
resource test {
    on hasta {
        local /dev/ad6
        remote 172.16.0.2
    }
    on hastb {
        local /dev/ad6
        remote 172.16.0.1
    }
}
```

For more advanced configuration, refer to hast.conf(5).

It is also possible to use host names in the `remote` statements if the hosts are resolvable and defined either in /etc/hosts or in the local DNS.

Once the configuration exists on both nodes, the HAST pool can be created. Run these commands on both nodes to place the initial metadata onto the local disk and to start hastd(8):

```
# hastctl create test
# service hastd onestart
```

It is *not* possible to use GEOM providers with an existing file system or to convert an existing storage to a HAST-managed pool. This procedure needs to store some metadata on the provider and there will not be enough required space available on an existing provider.

A HAST node's `primary` or `secondary` role is selected by an administrator, or software like Heartbeat, using hastctl(8). On the primary node, `hasta`, issue this command:

```
# hastctl role primary test
```

Run this command on the secondary node, `hastb`:

```
# hastctl role secondary test
```

Verify the result by running `hastctl` on each node:

```
# hastctl status test
```

Check the `status` line in the output. If it says `degraded`, something is wrong with the configuration file. It should say `complete` on each node, meaning that the synchronization between the nodes has started. The synchronization completes when `hastctl status` reports 0 bytes of `dirty` extents.

The next step is to create a file system on the GEOM provider and mount it. This must be done on the `primary` node. Creating the file system can take a few minutes, depending on the size of the hard drive. This example creates a UFS file system on /dev/hast/test:

```
# newfs -U /dev/hast/test
# mkdir /hast/test
# mount /dev/hast/test /hast/test
```

Once the HAST framework is configured properly, the final step is to make sure that HAST is started automatically during system boot. Add this line to /etc/rc.conf:

```
hastd_enable="YES"
```

Failover Configuration

The goal of this example is to build a robust storage system which is resistant to the failure of any given node. If the primary node fails, the secondary node is there to take over seamlessly, check and mount the file system, and continue to work without missing a single bit of data.

To accomplish this task, the Common Address Redundancy Protocol (CARP) is used to provide for automatic failover at the IP layer. CARP allows multiple hosts on the same network segment to share an IP address. Set up CARP on both nodes of the cluster according to the documentation available in "Common Address Redundancy Protocol (CARP)". In this example, each node will have its own management IP address and a shared IP address of *172.16.0.254*. The primary HAST node of the cluster must be the primary CARP node.

The HAST pool created in the previous section is now ready to be exported to the other hosts on the network. This can be accomplished by exporting it through NFS or Samba, using the shared IP address *172.16.0.254*. The only problem which remains unresolved is an automatic failover should the primary node fail.

In the event of CARP interfaces going up or down, the FreeBSD operating system generates a devd(8) event, making it possible to watch for state changes on the CARP interfaces. A state change on the CARP interface is an indication that one of the nodes failed or came back online. These state change events make it possible to run a script which will automatically handle the HAST failover.

To catch state changes on the CARP interfaces, add this configuration to /etc/devd.conf on each node, while replacing `<vhid>` with the virtual host id and `<ifname>` with the associated interface name:

```
notify 30 {
    match "system" "CARP";
    match "subsystem" "<vhid>@<ifname>";
    match "type" "MASTER";
    action "/usr/local/sbin/carp-hast-switch primary";
};

notify 30 {
    match "system" "CARP";
    match "subsystem" "<vhid>@<ifname>";
    match "type" "BACKUP";
    action "/usr/local/sbin/carp-hast-switch secondary";
};
```

Restart devd(8) on both nodes to put the new configuration into effect:

```
# service devd restart
```

When the specified interface state changes by going up or down , the system generates a notification, allowing the devd(8) subsystem to run the specified automatic failover script, /usr/local/sbin/carp-hast-switch. For further clarification about this configuration, refer to devd.conf(5).

Here is an example of an automated failover script:

```
#!/bin/sh

# Original script by Freddie Cash <fjwcash@gmail.com>
# Modified by Michael W. Lucas <mwlucas@BlackHelicopters.org>
# and Viktor Petersson <vpetersson@wireload.net>

# The names of the HAST resources, as listed in /etc/hast.conf
resources="test"

# delay in mounting HAST resource after becoming primary
# make your best guess
delay=3

# logging
log="local0.debug"
name="carp-hast"

# end of user configurable stuff

case "$1" in
    primary)
        logger -p $log -t $name "Switching to primary provider for ${resources}."
        sleep ${delay}
        # Wait for any "hastd secondary" processes to stop
        for disk in ${resources}; do
            while $( pgrep -lf "hastd: ${disk} \(secondary\)" > /dev/null 2>&1 ); do
                sleep 1
            done

            # Switch role for each disk
            hastctl role primary ${disk}
            if [ $? -ne 0 ]; then
                logger -p $log -t $name "Unable to change role to primary for resource
${disk}."
                exit 1
            fi
        done

        # Wait for the /dev/hast/* devices to appear
        for disk in ${resources}; do
            for I in $( jot 60 ); do
                [ -c "/dev/hast/${disk}" ] && break
                sleep 0.5
            done

            if [ ! -c "/dev/hast/${disk}" ]; then
                logger -p $log -t $name "GEOM provider /dev/hast/${disk} did not
appear."
                exit 1
            fi
        done

        logger -p $log -t $name "Role for HAST resources ${resources} switched to
primary."

        logger -p $log -t $name "Mounting disks."
        for disk in ${resources}; do
            mkdir -p /hast/${disk}
            fsck -p -y -t ufs /dev/hast/${disk}
            mount /dev/hast/${disk} /hast/${disk}
        done

    ;;

    secondary)
        logger -p $log -t $name "Switching to secondary provider for ${resources}."

        # Switch roles for the HAST resources
        for disk in ${resources}; do
            if ! mount | grep -q "^/dev/hast/${disk} on "
            then
            else
                umount -f /hast/${disk}
            fi
            sleep $delay
            hastctl role secondary ${disk} 2>&1
            if [ $? -ne 0 ]; then
                logger -p $log -t $name "Unable to switch role to secondary for
resource ${disk}."
                exit 1
            fi
            logger -p $log -t $name "Role switched to secondary for resource ${disk}."
        done
    ;;
esac
```

In a nutshell, the script takes these actions when a node becomes primary:

- Promotes the HAST pool to primary on the other node.
- Checks the file system under the HAST pool.
- Mounts the pool.

When a node becomes secondary:

- Unmounts the HAST pool.
- Degrades the HAST pool to secondary.

This is just an example script which serves as a proof of concept. It does not handle all the possible scenarios and can be extended or altered in any way, for example, to start or stop required services.

For this example, a standard UFS file system was used. To reduce the time needed for recovery, a journal-enabled UFS or ZFS file system can be used instead.

Instead of using the highly available storage locally, it can also be shared to other computers on a network via NFS, iSCSI, sshfs(1), or programs in ports (i.e. net/samba419).

More detailed information with additional examples can be found at http://wiki.FreeBSD.org/HAST.

21.14.3. Troubleshooting

HAST should generally work without issues. However, as with any other software product, there may be times when it does not work as supposed. The sources of the problems may be different, but the rule of thumb is to ensure that the time is synchronized between the nodes of the cluster.

When troubleshooting HAST, the debugging level of hastd(8) should be increased by starting `hastd` with `-d`. This argument may be specified multiple times to further increase the debugging level. Consider also using `-F`, which starts `hastd` in the foreground.

Recovering from the Split-brain Condition

Split-brain occurs when the nodes of the cluster are unable to communicate with each other, and both are configured as primary. This is a dangerous condition because it allows both nodes to make incompatible changes to the data. This problem must be corrected manually by the system administrator.

The administrator must either decide which node has more important changes, or perform the merge manually. Then, let HAST perform full synchronization of the node which has the broken data. To do this, issue these commands on the node which needs to be resynchronized:

```
# hastctl role init test
# hastctl create test
# hastctl role secondary test
```

Chapter 22. GEOM: Modular Disk Transformation Framework

22.1. Synopsis

In FreeBSD, the GEOM framework permits access and control to classes, such as Master Boot Records and BSD labels, through the use of providers, or the disk devices in /dev. By supporting various software RAID configurations, GEOM transparently provides access to the operating system and operating system utilities.

This chapter covers the use of disks under the GEOM framework in FreeBSD. This includes the major RAID control utilities which use the framework for configuration. This chapter is not a definitive guide to RAID configurations and only GEOM-supported RAID classifications are discussed.

Read this chapter to learn:

- What type of RAID support is available through GEOM.
- How to use the base utilities to configure, maintain, and manipulate the various RAID levels.
- How to mirror, stripe, encrypt, and remotely connect disk devices through GEOM.
- How to troubleshoot disks attached to the GEOM framework.

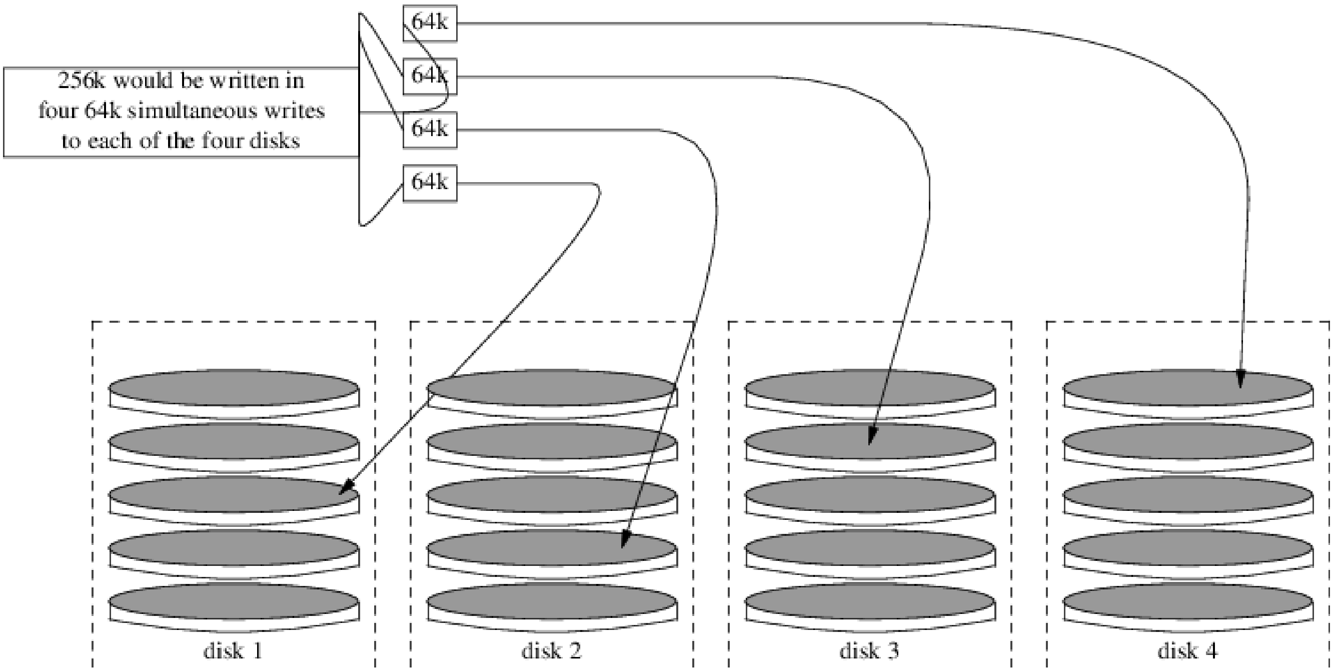
Before reading this chapter:

- Understand how FreeBSD treats disk devices (Storage).
- Know how to configure and install a new kernel (Configuring the FreeBSD Kernel).

22.2. RAID0 - Striping

Striping combines several disk drives into a single volume. Striping can be performed through the use of hardware RAID controllers. The GEOM disk subsystem provides software support for disk striping, also known as RAID0, without the need for a RAID disk controller.

In RAID0, data is split into blocks that are written across all the drives in the array. As seen in the following illustration, instead of having to wait on the system to write 256k to one disk, RAID0 can simultaneously write 64k to each of the four disks in the array, offering superior I/O performance. This performance can be enhanced further by using multiple disk controllers.


256k would be written in four 64k simultaneous writes to each of the four disks
64k
64k
64k
64k
disk 1
disk 2
disk 3
disk 4

Each disk in a RAID0 stripe must be of the same size, since I/O requests are interleaved to read or write to multiple disks in parallel.

RAID0 does *not* provide any redundancy. This means that if one disk in the array fails, all of the data on the disks is lost. If the data is important, implement a backup strategy that regularly saves backups to a remote system or device.

The process for creating a software, GEOM-based RAID0 on a FreeBSD system using commodity disks is as follows. Once the stripe is created, refer to gstripe(8) for more information on how to control an existing stripe.

Procedure: Creating a Stripe of Unformatted ATA Disks

1. Load the geom_stripe.ko module:

```
# kldload geom_stripe
```

2. Ensure that a suitable mount point exists. If this volume will become a root partition, then temporarily use another mount point such as /mnt.
3. Determine the device names for the disks which will be striped, and create the new stripe device. For example, to stripe two unused and unpartitioned ATA disks with device names of /dev/ad2 and /dev/ad3:

```
# gstripe label -v st0 /dev/ad2 /dev/ad3
Metadata value stored on /dev/ad2.
Metadata value stored on /dev/ad3.
Done.
```

4. Write a standard label, also known as a partition table, on the new volume and install the

default bootstrap code:

```
# bsdlabel -wB /dev/stripe/st0
```

5. This process should create two other devices in /dev/stripe in addition to st0. Those include st0a and st0c. At this point, a UFS file system can be created on st0a using `newfs`:

   ```
   # newfs -U /dev/stripe/st0a
   ```

   Many numbers will glide across the screen, and after a few seconds, the process will be complete. The volume has been created and is ready to be mounted.

6. To manually mount the created disk stripe:

   ```
   # mount /dev/stripe/st0a /mnt
   ```

7. To mount this striped file system automatically during the boot process, place the volume information in /etc/fstab. In this example, a permanent mount point, named stripe, is created:

   ```
   # mkdir /stripe
   # echo "/dev/stripe/st0a /stripe ufs rw 2 2" \
   >> /etc/fstab
   ```

8. The geom_stripe.ko module must also be automatically loaded during system initialization, by adding a line to /boot/loader.conf:

   ```
   # echo 'geom_stripe_load="YES"' >> /boot/loader.conf
   ```

22.3. RAID1 - Mirroring

RAID1, or *mirroring*, is the technique of writing the same data to more than one disk drive. Mirrors are usually used to guard against data loss due to drive failure. Each drive in a mirror contains an identical copy of the data. When an individual drive fails, the mirror continues to work, providing data from the drives that are still functioning. The computer keeps running, and the administrator has time to replace the failed drive without user interruption.

Two common situations are illustrated in these examples. The first creates a mirror out of two new drives and uses it as a replacement for an existing single drive. The second example creates a mirror on a single new drive, copies the old drive's data to it, then inserts the old drive into the mirror. While this procedure is slightly more complicated, it only requires one new drive.

Traditionally, the two drives in a mirror are identical in model and capacity, but gmirror(8) does not require that. Mirrors created with dissimilar drives will have a capacity equal to that of the smallest drive in the mirror. Extra space on larger drives will be unused. Drives inserted into the mirror later must have at least as much capacity as the smallest drive already in the mirror.

The mirroring procedures shown here are non-destructive, but as with any major disk operation, make a full backup first.

While dump(8) is used in these procedures to copy file systems, it does not work on file systems with soft updates journaling. See tunefs(8) for information on detecting and disabling soft updates journaling.

22.3.1. Metadata Issues

Many disk systems store metadata at the end of each disk. Old metadata should be erased before reusing the disk for a mirror. Most problems are caused by two particular types of leftover metadata: GPT partition tables and old metadata from a previous mirror.

GPT metadata can be erased with gpart(8). This example erases both primary and backup GPT partition tables from disk ada8:

```
# gpart destroy -F ada8
```

A disk can be removed from an active mirror and the metadata erased in one step using gmirror(8). Here, the example disk ada8 is removed from the active mirror gm4:

```
# gmirror remove gm4 ada8
```

If the mirror is not running, but old mirror metadata is still on the disk, use `gmirror clear` to remove it:

```
# gmirror clear ada8
```

gmirror(8) stores one block of metadata at the end of the disk. As GPT partition schemes also store metadata at the end of the disk, mirroring entire GPT disks with gmirror(8) is not recommended. MBR partitioning is used here because it only stores a partition table at the start of the disk and does not conflict with the mirror metadata.

22.3.2. Creating a Mirror with Two New Disks

In this example, FreeBSD has already been installed on a single disk, ada0. Two new disks, ada1 and ada2, have been connected to the system. A new mirror will be created on these two disks and used to replace the old single disk.

The geom_mirror.ko kernel module must either be built into the kernel or loaded at boot- or run-time. Manually load the kernel module now:

```
# gmirror load
```

Create the mirror with the two new drives:

```
# gmirror label -v gm0 /dev/ada1 /dev/ada2
```

gm0 is a user-chosen device name assigned to the new mirror. After the mirror has been started, this device name appears in /dev/mirror/.

MBR and bsdlabel partition tables can now be created on the mirror with gpart(8). This example uses a traditional file system layout, with partitions for /, swap, /var, /tmp, and /usr. A single / and a swap partition will also work.

Partitions on the mirror do not have to be the same size as those on the existing disk, but they must be large enough to hold all the data already present on ada0.

```
# gpart create -s MBR mirror/gm0
# gpart add -t freebsd -a 4k mirror/gm0
# gpart show mirror/gm0
=>       63  156301423  mirror/gm0  MBR  (74G)
         63         63                    - free -  (31k)
        126  156301299                1  freebsd  (74G)
  156301425         61                    - free -  (30k)
```

```
# gpart create -s BSD mirror/gm0s1
# gpart add -t freebsd-ufs  -a 4k -s 2g mirror/gm0s1
# gpart add -t freebsd-swap -a 4k -s 4g mirror/gm0s1
# gpart add -t freebsd-ufs  -a 4k -s 2g mirror/gm0s1
# gpart add -t freebsd-ufs  -a 4k -s 1g mirror/gm0s1
# gpart add -t freebsd-ufs  -a 4k mirror/gm0s1
# gpart show mirror/gm0s1
=>        0  156301299  mirror/gm0s1  BSD  (74G)
          0          2                      - free -  (1.0k)
          2    4194304                  1  freebsd-ufs  (2.0G)
    4194306    8388608                  2  freebsd-swap (4.0G)
   12582914    4194304                  4  freebsd-ufs  (2.0G)
   16777218    2097152                  5  freebsd-ufs  (1.0G)
   18874370  137426928                  6  freebsd-ufs  (65G)
  156301298          1                      - free -  (512B)
```

Make the mirror bootable by installing bootcode in the MBR and bsdlabel and setting the active slice:

```
# gpart bootcode -b /boot/mbr mirror/gm0
# gpart set -a active -i 1 mirror/gm0
# gpart bootcode -b /boot/boot mirror/gm0s1
```

Format the file systems on the new mirror, enabling soft-updates.

```
# newfs -U /dev/mirror/gm0s1a
# newfs -U /dev/mirror/gm0s1d
# newfs -U /dev/mirror/gm0s1e
# newfs -U /dev/mirror/gm0s1f
```

File systems from the original ada0 disk can now be copied onto the mirror with dump(8) and restore(8).

```
# mount /dev/mirror/gm0s1a /mnt
# dump -C16 -b64 -0aL -f - / | (cd /mnt && restore -rf -)
# mount /dev/mirror/gm0s1d /mnt/var
# mount /dev/mirror/gm0s1e /mnt/tmp
# mount /dev/mirror/gm0s1f /mnt/usr
# dump -C16 -b64 -0aL -f - /var | (cd /mnt/var && restore -rf -)
# dump -C16 -b64 -0aL -f - /tmp | (cd /mnt/tmp && restore -rf -)
# dump -C16 -b64 -0aL -f - /usr | (cd /mnt/usr && restore -rf -)
```

Edit /mnt/etc/fstab to point to the new mirror file systems:

```
# Device        Mountpoint  FStype  Options Dump    Pass#
/dev/mirror/gm0s1a  /       ufs rw  1   1
/dev/mirror/gm0s1b  none        swap    sw  0   0
/dev/mirror/gm0s1d  /var        ufs rw  2   2
/dev/mirror/gm0s1e  /tmp        ufs rw  2   2
/dev/mirror/gm0s1f  /usr        ufs rw  2   2
```

If the geom_mirror.ko kernel module has not been built into the kernel, /mnt/boot/loader.conf is edited to load the module at boot:

```
geom_mirror_load="YES"
```

Reboot the system to test the new mirror and verify that all data has been copied. The BIOS will see the mirror as two individual drives rather than a mirror. Since the drives are identical, it does not matter which is selected to boot.

See Troubleshooting if there are problems booting. Powering down and disconnecting the original ada0 disk will allow it to be kept as an offline backup.

In use, the mirror will behave just like the original single drive.

22.3.3. Creating a Mirror with an Existing Drive

In this example, FreeBSD has already been installed on a single disk, ada0. A new disk, ada1, has been connected to the system. A one-disk mirror will be created on the new disk, the existing system copied onto it, and then the old disk will be inserted into the mirror. This slightly complex procedure is required because `gmirror` needs to put a 512-byte block of metadata at the end of each disk, and the existing ada0 has usually had all of its space already allocated.

Load the geom_mirror.ko kernel module:

```
# gmirror load
```

Check the media size of the original disk with `diskinfo`:

```
# diskinfo -v ada0 | head -n3
/dev/ada0
        512             # sectorsize
        1000204821504   # mediasize in bytes (931G)
```

Create a mirror on the new disk. To make certain that the mirror capacity is not any larger than the original ada0 drive, gnop(8) is used to create a fake drive of the same size. This drive does not store any data, but is used only to limit the size of the mirror. When gmirror(8) creates the mirror, it will restrict the capacity to the size of gzero.nop, even if the new ada1 drive has more space. Note that the *1000204821504* in the second line is equal to ada0's media size as shown by `diskinfo` above.

```
# geom zero load
# gnop create -s 1000204821504 gzero
# gmirror label -v gm0 gzero.nop ada1
# gmirror forget gm0
```

Since gzero.nop does not store any data, the mirror does not see it as connected. The mirror is told to "forget" unconnected components, removing references to gzero.nop. The result is a mirror device containing only a single disk, ada1.

After creating gm0, view the partition table on ada0. This output is from a 1 TB drive. If there is some unallocated space at the end of the drive, the contents may be copied directly from ada0 to the new mirror.

However, if the output shows that all of the space on the disk is allocated, as in the following listing, there is no space available for the 512-byte mirror metadata at the end of the disk.

```
# gpart show ada0
=>        63  1953525105        ada0  MBR  (931G)
          63  1953525105           1  freebsd  [active]  (931G)
```

In this case, the partition table must be edited to reduce the capacity by one sector on mirror/gm0.

The procedure will be explained later.

In either case, partition tables on the primary disk should be first copied using `gpart backup` and `gpart restore`.

```
# gpart backup ada0 > table.ada0
# gpart backup ada0s1 > table.ada0s1
```

These commands create two files, table.ada0 and table.ada0s1. This example is from a 1 TB drive:

```
# cat table.ada0
MBR 4
1 freebsd         63 1953525105   [active]
```

```
# cat table.ada0s1
BSD 8
1  freebsd-ufs          0    4194304
2 freebsd-swap    4194304   33554432
4  freebsd-ufs   37748736   50331648
5  freebsd-ufs   88080384   41943040
6  freebsd-ufs  130023424  838860800
7  freebsd-ufs  968884224  984640881
```

If no free space is shown at the end of the disk, the size of both the slice and the last partition must be reduced by one sector. Edit the two files, reducing the size of both the slice and last partition by one. These are the last numbers in each listing.

```
# cat table.ada0
MBR 4
1 freebsd         63 1953525104   [active]
```

```
# cat table.ada0s1
BSD 8
1  freebsd-ufs          0    4194304
2 freebsd-swap    4194304   33554432
4  freebsd-ufs   37748736   50331648
5  freebsd-ufs   88080384   41943040
6  freebsd-ufs  130023424  838860800
7  freebsd-ufs  968884224  984640880
```

If at least one sector was unallocated at the end of the disk, these two files can be used without modification.

Now restore the partition table into mirror/gm0:

```
# gpart restore mirror/gm0 < table.ada0
# gpart restore mirror/gm0s1 < table.ada0s1
```

Check the partition table with `gpart show`. This example has gm0s1a for /, gm0s1d for /var, gm0s1e for /usr, gm0s1f for /data1, and gm0s1g for /data2.

```
# gpart show mirror/gm0
=>        63  1953525104  mirror/gm0  MBR  (931G)
          63  1953525042           1  freebsd  [active]  (931G)
  1953525105          62              - free -  (31k)

# gpart show mirror/gm0s1
=>         0  1953525042  mirror/gm0s1  BSD  (931G)
           0     2097152             1  freebsd-ufs  (1.0G)
     2097152    16777216             2  freebsd-swap  (8.0G)
    18874368    41943040             4  freebsd-ufs  (20G)
    60817408    20971520             5  freebsd-ufs  (10G)
    81788928   629145600             6  freebsd-ufs  (300G)
   710934528  1242590514             7  freebsd-ufs  (592G)
  1953525042          63                - free -  (31k)
```

Both the slice and the last partition must have at least one free block at the end of the disk.

Create file systems on these new partitions. The number of partitions will vary to match the original disk, ada0.

```
# newfs -U /dev/mirror/gm0s1a
# newfs -U /dev/mirror/gm0s1d
# newfs -U /dev/mirror/gm0s1e
# newfs -U /dev/mirror/gm0s1f
# newfs -U /dev/mirror/gm0s1g
```

Make the mirror bootable by installing bootcode in the MBR and bsdlabel and setting the active slice:

```
# gpart bootcode -b /boot/mbr mirror/gm0
# gpart set -a active -i 1 mirror/gm0
# gpart bootcode -b /boot/boot mirror/gm0s1
```

Adjust /etc/fstab to use the new partitions on the mirror. Back up this file first by copying it to /etc/fstab.orig.

```
# cp /etc/fstab /etc/fstab.orig
```

Edit /etc/fstab, replacing /dev/ada0 with mirror/gm0.

```
# Device        Mountpoint  FStype  Options Dump    Pass#
/dev/mirror/gm0s1a  /       ufs rw  1   1
/dev/mirror/gm0s1b  none        swap    sw  0   0
/dev/mirror/gm0s1d  /var        ufs rw  2   2
/dev/mirror/gm0s1e  /usr        ufs rw  2   2
/dev/mirror/gm0s1f  /data1      ufs rw  2   2
/dev/mirror/gm0s1g  /data2      ufs rw  2   2
```

If the geom_mirror.ko kernel module has not been built into the kernel, edit /boot/loader.conf to load it at boot:

```
geom_mirror_load="YES"
```

File systems from the original disk can now be copied onto the mirror with dump(8) and restore(8). Each file system dumped with `dump -L` will create a snapshot first, which can take some time.

```
# mount /dev/mirror/gm0s1a /mnt
# dump -C16 -b64 -0aL -f - /    | (cd /mnt && restore -rf -)
# mount /dev/mirror/gm0s1d /mnt/var
# mount /dev/mirror/gm0s1e /mnt/usr
# mount /dev/mirror/gm0s1f /mnt/data1
# mount /dev/mirror/gm0s1g /mnt/data2
# dump -C16 -b64 -0aL -f - /usr | (cd /mnt/usr && restore -rf -)
# dump -C16 -b64 -0aL -f - /var | (cd /mnt/var && restore -rf -)
# dump -C16 -b64 -0aL -f - /data1 | (cd /mnt/data1 && restore -rf -)
# dump -C16 -b64 -0aL -f - /data2 | (cd /mnt/data2 && restore -rf -)
```

Restart the system, booting from ada1. If everything is working, the system will boot from mirror/gm0, which now contains the same data as ada0 had previously. See Troubleshooting if there are problems booting.

At this point, the mirror still consists of only the single ada1 disk.

After booting from mirror/gm0 successfully, the final step is inserting ada0 into the mirror.

> When ada0 is inserted into the mirror, its former contents will be overwritten by data from the mirror. Make certain that mirror/gm0 has the same contents as ada0 before adding ada0 to the mirror. If the contents previously copied by dump(8) and restore(8) are not identical to what was on ada0, revert /etc/fstab to mount the file systems on ada0, reboot, and start the whole procedure again.

```
# gmirror insert gm0 ada0
GEOM_MIRROR: Device gm0: rebuilding provider ada0
```

Synchronization between the two disks will start immediately. Use `gmirror status` to view the progress.

```
# gmirror status
      Name    Status  Components
mirror/gm0  DEGRADED  ada1 (ACTIVE)
                      ada0 (SYNCHRONIZING, 64%)
```

After a while, synchronization will finish.

```
GEOM_MIRROR: Device gm0: rebuilding provider ada0 finished.
# gmirror status
      Name    Status  Components
mirror/gm0  COMPLETE  ada1 (ACTIVE)
                      ada0 (ACTIVE)
```

mirror/gm0 now consists of the two disks ada0 and ada1, and the contents are automatically synchronized with each other. In use, mirror/gm0 will behave just like the original single drive.

22.3.4. Troubleshooting

If the system no longer boots, BIOS settings may have to be changed to boot from one of the new mirrored drives. Either mirror drive can be used for booting, as they contain identical data.

If the boot stops with this message, something is wrong with the mirror device:

```
Mounting from ufs:/dev/mirror/gm0s1a failed with error 19.

Loader variables:
  vfs.root.mountfrom=ufs:/dev/mirror/gm0s1a
  vfs.root.mountfrom.options=rw

Manual root filesystem specification:
  <fstype>:<device> [options]
      Mount <device> using filesystem <fstype>
      and with the specified (optional) option list.

    e.g. ufs:/dev/da0s1a
        zfs:tank
        cd9660:/dev/acd0 ro
          (which is equivalent to: mount -t cd9660 -o ro /dev/acd0 /)

  ?               List valid disk boot devices
  .               Yield 1 second (for background tasks)
  <empty line>    Abort manual input

mountroot>
```

Forgetting to load the geom_mirror.ko module in /boot/loader.conf can cause this problem. To fix it, boot from a FreeBSD installation media and choose `Shell` at the first prompt. Then load the mirror module and mount the mirror device:

```
# gmirror load
# mount /dev/mirror/gm0s1a /mnt
```

Edit /mnt/boot/loader.conf, adding a line to load the mirror module:

```
geom_mirror_load="YES"
```

Save the file and reboot.

Other problems that cause `error 19` require more effort to fix. Although the system should boot from ada0, another prompt to select a shell will appear if /etc/fstab is incorrect. Enter `ufs:/dev/ada0s1a` at the boot loader prompt and press `Enter`. Undo the edits in /etc/fstab then mount the file systems from the original disk (ada0) instead of the mirror. Reboot the system and try the procedure again.

```
Enter full pathname of shell or RETURN for /bin/sh:
# cp /etc/fstab.orig /etc/fstab
# shutdown -r now
```

22.3.5. Recovering from Disk Failure

The benefit of disk mirroring is that an individual disk can fail without causing the mirror to lose any data. In the above example, if ada0 fails, the mirror will continue to work, providing data from the remaining working drive, ada1.

To replace the failed drive, shut down the system and physically replace the failed drive with a new drive of equal or greater capacity. Manufacturers use somewhat arbitrary values when rating drives in gigabytes, and the only way to really be sure is to compare the total count of sectors shown by `diskinfo -v`. A drive with larger capacity than the mirror will work, although the extra space on the new drive will not be used.

After the computer is powered back up, the mirror will be running in a "degraded" mode with only one drive. The mirror is told to forget drives that are not currently connected:

```
# gmirror forget gm0
```

Any old metadata should be cleared from the replacement disk using the instructions in Metadata Issues. Then the replacement disk, ada4 for this example, is inserted into the mirror:

```
# gmirror insert gm0 /dev/ada4
```

Resynchronization begins when the new drive is inserted into the mirror. This process of copying mirror data to a new drive can take a while. Performance of the mirror will be greatly reduced during the copy, so inserting new drives is best done when there is low demand on the computer.

Progress can be monitored with `gmirror status`, which shows drives that are being synchronized and the percentage of completion. During resynchronization, the status will be `DEGRADED`, changing to `COMPLETE` when the process is finished.

22.4. RAID3 - Byte-level Striping with Dedicated Parity

RAID3 is a method used to combine several disk drives into a single volume with a dedicated parity disk. In a RAID3 system, data is split up into a number of bytes that are written across all the drives in the array except for one disk which acts as a dedicated parity disk. This means that disk reads from a RAID3 implementation access all disks in the array. Performance can be enhanced by using multiple disk controllers. The RAID3 array provides a fault tolerance of 1 drive, while providing a capacity of 1 - 1/n times the total capacity of all drives in the array, where n is the number of hard drives in the array. Such a configuration is mostly suitable for storing data of larger sizes such as multimedia files.

At least 3 physical hard drives are required to build a RAID3 array. Each disk must be of the same size, since I/O requests are interleaved to read or write to multiple disks in parallel. Also, due to the nature of RAID3, the number of drives must be equal to 3, 5, 9, 17, and so on, or 2^n + 1.

This section demonstrates how to create a software RAID3 on a FreeBSD system.

> While it is theoretically possible to boot from a RAID3 array on FreeBSD, that configuration is uncommon and is not advised.

22.4.1. Creating a Dedicated RAID3 Array

In FreeBSD, support for RAID3 is implemented by the graid3(8) GEOM class. Creating a dedicated RAID3 array on FreeBSD requires the following steps.

1. First, load the geom_raid3.ko kernel module by issuing one of the following commands:

   ```
   # graid3 load
   ```

   or:

   ```
   # kldload geom_raid3
   ```

2. Ensure that a suitable mount point exists. This command creates a new directory to use as the mount point:

   ```
   # mkdir /multimedia
   ```

3. Determine the device names for the disks which will be added to the array, and create the new RAID3 device. The final device listed will act as the dedicated parity disk. This example uses three unpartitioned ATA drives: ada1 and ada2 for data, and ada3 for parity.

```
# graid3 label -v gr0 /dev/ada1 /dev/ada2 /dev/ada3
Metadata value stored on /dev/ada1.
Metadata value stored on /dev/ada2.
Metadata value stored on /dev/ada3.
Done.
```

4. Partition the newly created gr0 device and put a UFS file system on it:

```
# gpart create -s GPT /dev/raid3/gr0
# gpart add -t freebsd-ufs /dev/raid3/gr0
# newfs -j /dev/raid3/gr0p1
```

   Many numbers will glide across the screen, and after a bit of time, the process will be complete. The volume has been created and is ready to be mounted:

```
# mount /dev/raid3/gr0p1 /multimedia/
```

   The RAID3 array is now ready to use.

Additional configuration is needed to retain this setup across system reboots.

1. The geom_raid3.ko module must be loaded before the array can be mounted. To automatically load the kernel module during system initialization, add the following line to /boot/loader.conf:

```
geom_raid3_load="YES"
```

2. The following volume information must be added to /etc/fstab in order to automatically mount the array's file system during the system boot process:

```
/dev/raid3/gr0p1    /multimedia ufs rw  2   2
```

22.5. Software RAID Devices

Some motherboards and expansion cards add some simple hardware, usually just a ROM, that allows the computer to boot from a RAID array. After booting, access to the RAID array is handled by software running on the computer's main processor. This "hardware-assisted software RAID" gives RAID arrays that are not dependent on any particular operating system, and which are functional even before an operating system is loaded.

Several levels of RAID are supported, depending on the hardware in use. See graid(8) for a complete list.

graid(8) requires the geom_raid.ko kernel module, which is included in the GENERIC kernel starting with FreeBSD 9.1. If needed, it can be loaded manually with `graid load`.

22.5.1. Creating an Array

Software RAID devices often have a menu that can be entered by pressing special keys when the computer is booting. The menu can be used to create and delete RAID arrays. graid(8) can also create arrays directly from the command line.

`graid label` is used to create a new array. The motherboard used for this example has an Intel software RAID chipset, so the Intel metadata format is specified. The new array is given a label of gm0, it is a mirror (RAID1), and uses drives ada0 and ada1.

Some space on the drives will be overwritten when they are made into a new array. Back up existing data first!

```
# graid label Intel gm0 RAID1 ada0 ada1
GEOM_RAID: Intel-a29ea104: Array Intel-a29ea104 created.
GEOM_RAID: Intel-a29ea104: Disk ada0 state changed from NONE to ACTIVE.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:0-ada0 state changed from NONE to ACTIVE.
GEOM_RAID: Intel-a29ea104: Disk ada1 state changed from NONE to ACTIVE.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:1-ada1 state changed from NONE to ACTIVE.
GEOM_RAID: Intel-a29ea104: Array started.
GEOM_RAID: Intel-a29ea104: Volume gm0 state changed from STARTING to OPTIMAL.
Intel-a29ea104 created
GEOM_RAID: Intel-a29ea104: Provider raid/r0 for volume gm0 created.
```

A status check shows the new mirror is ready for use:

```
# graid status
   Name   Status  Components
raid/r0  OPTIMAL  ada0 (ACTIVE (ACTIVE))
                  ada1 (ACTIVE (ACTIVE))
```

The array device appears in /dev/raid/. The first array is called r0. Additional arrays, if present, will be r1, r2, and so on.

The BIOS menu on some of these devices can create arrays with special characters in their names. To avoid problems with those special characters, arrays are given simple numbered names like r0. To show the actual labels, like gm0 in the example above, use sysctl(8):

```
# sysctl kern.geom.raid.name_format=1
```

22.5.2. Multiple Volumes

Some software RAID devices support more than one *volume* on an array. Volumes work like partitions, allowing space on the physical drives to be split and used in different ways. For example, Intel software RAID devices support two volumes. This example creates a 40 G mirror for safely storing the operating system, followed by a 20 G RAID0 (stripe) volume for fast temporary storage:

```
# graid label -S 40G Intel gm0 RAID1 ada0 ada1
# graid add -S 20G gm0 RAID0
```

Volumes appear as additional rX entries in /dev/raid/. An array with two volumes will show r0 and r1.

See graid(8) for the number of volumes supported by different software RAID devices.

22.5.3. Converting a Single Drive to a Mirror

Under certain specific conditions, it is possible to convert an existing single drive to a graid(8) array without reformatting. To avoid data loss during the conversion, the existing drive must meet these minimum requirements:

- The drive must be partitioned with the MBR partitioning scheme. GPT or other partitioning schemes with metadata at the end of the drive will be overwritten and corrupted by the graid(8) metadata.
- There must be enough unpartitioned and unused space at the end of the drive to hold the graid(8) metadata. This metadata varies in size, but the largest occupies 64 M, so at least that much free space is recommended.

If the drive meets these requirements, start by making a full backup. Then create a single-drive mirror with that drive:

```
# graid label Intel gm0 RAID1 ada0 NONE
```

graid(8) metadata was written to the end of the drive in the unused space. A second drive can now be inserted into the mirror:

```
# graid insert raid/r0 ada1
```

Data from the original drive will immediately begin to be copied to the second drive. The mirror will operate in degraded status until the copy is complete.

22.5.4. Inserting New Drives into the Array

Drives can be inserted into an array as replacements for drives that have failed or are missing. If there are no failed or missing drives, the new drive becomes a spare. For example, inserting a new drive into a working two-drive mirror results in a two-drive mirror with one spare drive, not a three-drive mirror.

In the example mirror array, data immediately begins to be copied to the newly-inserted drive. Any existing information on the new drive will be overwritten.

```
# graid insert raid/r0 ada1
GEOM_RAID: Intel-a29ea104: Disk ada1 state changed from NONE to ACTIVE.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:1-ada1 state changed from NONE to NEW.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:1-ada1 state changed from NEW to REBUILD.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:1-ada1 rebuild start at 0.
```

22.5.5. Removing Drives from the Array

Individual drives can be permanently removed from a from an array and their metadata erased:

```
# graid remove raid/r0 ada1
GEOM_RAID: Intel-a29ea104: Disk ada1 state changed from ACTIVE to OFFLINE.
GEOM_RAID: Intel-a29ea104: Subdisk gm0:1-[unknown] state changed from ACTIVE to NONE.
GEOM_RAID: Intel-a29ea104: Volume gm0 state changed from OPTIMAL to DEGRADED.
```

22.5.6. Stopping the Array

An array can be stopped without removing metadata from the drives. The array will be restarted when the system is booted.

```
# graid stop raid/r0
```

22.5.7. Checking Array Status

Array status can be checked at any time. After a drive was added to the mirror in the example above, data is being copied from the original drive to the new drive:

```
# graid status
   Name    Status  Components
raid/r0  DEGRADED  ada0 (ACTIVE (ACTIVE))
                   ada1 (ACTIVE (REBUILD 28%))
```

Some types of arrays, like `RAID0` or `CONCAT`, may not be shown in the status report if disks have failed. To see these partially-failed arrays, add `-ga`:

```
# graid status -ga
          Name  Status  Components
Intel-e2d07d9a  BROKEN  ada6 (ACTIVE (ACTIVE))
```

22.5.8. Deleting Arrays

Arrays are destroyed by deleting all of the volumes from them. When the last volume present is deleted, the array is stopped and metadata is removed from the drives:

```
# graid delete raid/r0
```

22.5.9. Deleting Unexpected Arrays

Drives may unexpectedly contain graid(8) metadata, either from previous use or manufacturer testing. graid(8) will detect these drives and create an array, interfering with access to the individual drive. To remove the unwanted metadata:

1. Boot the system. At the boot menu, select `2` for the loader prompt. Enter:

   ```
   OK set kern.geom.raid.enable=0
   OK boot
   ```

   The system will boot with graid(8) disabled.

2. Back up all data on the affected drive.
3. As a workaround, graid(8) array detection can be disabled by adding

   ```
   kern.geom.raid.enable=0
   ```

   to /boot/loader.conf.

   To permanently remove the graid(8) metadata from the affected drive, boot a FreeBSD installation CD-ROM or memory stick, and select `Shell`. Use `status` to find the name of the array, typically `raid/r0`:

   ```
   # graid status
      Name   Status  Components
   raid/r0  OPTIMAL  ada0 (ACTIVE (ACTIVE))
                     ada1 (ACTIVE (ACTIVE))
   ```

   Delete the volume by name:

   ```
   # graid delete raid/r0
   ```

   If there is more than one volume shown, repeat the process for each volume. After the last array has been deleted, the volume will be destroyed.

   Reboot and verify data, restoring from backup if necessary. After the metadata has been removed, the `kern.geom.raid.enable=0` entry in /boot/loader.conf can also be removed.

22.6. GEOM Gate Network

GEOM provides a simple mechanism for providing remote access to devices such as disks, CDs, and file systems through the use of the GEOM Gate network daemon, ggated. The system with the device runs the server daemon which handles requests made by clients using ggatec. The devices should not contain any sensitive data as the connection between the client and the server is not encrypted.

Similar to NFS, which is discussed in Network File System (NFS), ggated is configured using an exports file. This file specifies which systems are permitted to access the exported resources and what level of access they are offered. For example, to give the client `192.168.1.5` read and write access to the fourth slice on the first SCSI disk, create /etc/gg.exports with this line:

```
192.168.1.5 RW /dev/da0s4d
```

Before exporting the device, ensure it is not currently mounted. Then, start ggated:

```
# ggated
```

Several options are available for specifying an alternate listening port or changing the default location of the exports file. Refer to ggated(8) for details.

To access the exported device on the client machine, first use `ggatec` to specify the IP address of the server and the device name of the exported device. If successful, this command will display a `ggate` device name to mount. Mount that specified device name on a free mount point. This example connects to the /dev/da0s4d partition on `192.168.1.1`, then mounts /dev/ggate0 on /mnt:

```
# ggatec create -o rw 192.168.1.1 /dev/da0s4d
ggate0
# mount /dev/ggate0 /mnt
```

The device on the server may now be accessed through /mnt on the client. For more details about `ggatec` and a few usage examples, refer to ggatec(8).

The mount will fail if the device is currently mounted on either the server or any other client on the network. If simultaneous access is needed to network resources, use NFS instead.

When the device is no longer needed, unmount it with `umount` so that the resource is available to other clients.

22.7. Labeling Disk Devices

During system initialization, the FreeBSD kernel creates device nodes as devices are found. This method of probing for devices raises some issues. For instance, what if a new disk device is added via USB? It is likely that a flash device may be handed the device name of da0 and the original da0 shifted to da1. This will cause issues mounting file systems if they are listed in /etc/fstab which may also prevent the system from booting.

One solution is to chain SCSI devices in order so a new device added to the SCSI card will be issued unused device numbers. But what about USB devices which may replace the primary SCSI disk? This happens because USB devices are usually probed before the SCSI card. One solution is to only insert these devices after the system has been booted. Another method is to use only a single ATA drive and never list the SCSI devices in /etc/fstab.

A better solution is to use `glabel` to label the disk devices and use the labels in /etc/fstab. Since `glabel` stores the label in the last sector of a given provider, the label will remain persistent across reboots. By using this label as a device, the file-system may always be mounted regardless of what device node it is accessed through.

> `glabel` can create both transient and permanent labels. Only permanent labels are consistent across reboots. Refer to glabel(8) for more information on the differences between labels.

22.7.1. Label Types and Examples

Permanent labels can be a generic or a file system label. Permanent file system labels can be created with tunefs(8) or newfs(8). These types of labels are created in a sub-directory of /dev, and will be named according to the file system type. For example, UFS2 file system labels will be created in /dev/ufs. Generic permanent labels can be created with `glabel label`. These are not file system specific and will be created in /dev/label.

Temporary labels are destroyed at the next reboot. These labels are created in /dev/label and are suited to experimentation. A temporary label can be created using `glabel create`.

To create a permanent label for a UFS2 file system without destroying any data, issue the following command:

```
# tunefs -L home /dev/da3
```

A label should now exist in /dev/ufs which may be added to /etc/fstab:

```
/dev/ufs/home       /home            ufs     rw              2       2
```

> The file system must not be mounted while attempting to run `tunefs`.

Now the file system may be mounted:

```
# mount /home
```

From this point on, so long as the geom_label.ko kernel module is loaded at boot with /boot/loader.conf or the `GEOM_LABEL` kernel option is present, the device node may change without any ill effect on the system.

File systems may also be created with a default label by using the `-L` flag with `newfs`. Refer to newfs(8) for more information.

The following command can be used to destroy the label:

```
# glabel destroy home
```

The following example shows how to label the partitions of a boot disk.

Example 32. Labeling Partitions on the Boot Disk

By permanently labeling the partitions on the boot disk, the system should be able to continue to boot normally, even if the disk is moved to another controller or transferred to a different system. For this example, it is assumed that a single ATA disk is used, which is currently recognized by the system as ad0. It is also assumed that the standard FreeBSD partition scheme is used, with /, /var, /usr and /tmp, as well as a swap partition.

Reboot the system, and at the loader(8) prompt, press 4 to boot into single user mode. Then enter the following commands:

```
# glabel label rootfs /dev/ad0s1a
GEOM_LABEL: Label for provider /dev/ad0s1a is label/rootfs
# glabel label var /dev/ad0s1d
GEOM_LABEL: Label for provider /dev/ad0s1d is label/var
# glabel label usr /dev/ad0s1f
GEOM_LABEL: Label for provider /dev/ad0s1f is label/usr
# glabel label tmp /dev/ad0s1e
GEOM_LABEL: Label for provider /dev/ad0s1e is label/tmp
# glabel label swap /dev/ad0s1b
GEOM_LABEL: Label for provider /dev/ad0s1b is label/swap
# exit
```

The system will continue with multi-user boot. After the boot completes, edit /etc/fstab and replace the conventional device names, with their respective labels. The final /etc/fstab will look like this:

```
# Device                Mountpoint      FStype  Options         Dump    Pass#
/dev/label/swap         none            swap    sw              0       0
/dev/label/rootfs       /               ufs     rw              1       1
/dev/label/tmp          /tmp            ufs     rw              2       2
/dev/label/usr          /usr            ufs     rw              2       2
/dev/label/var          /var            ufs     rw              2       2
```

The system can now be rebooted. If everything went well, it will come up normally and `mount` will show:

```
# mount
/dev/label/rootfs on / (ufs, local)
devfs on /dev (devfs, local)
/dev/label/tmp on /tmp (ufs, local, soft-updates)
/dev/label/usr on /usr (ufs, local, soft-updates)
/dev/label/var on /var (ufs, local, soft-updates)
```

The glabel(8) class supports a label type for UFS file systems, based on the unique file system id, `ufsid`. These labels may be found in /dev/ufsid and are created automatically during system startup. It is possible to use `ufsid` labels to mount partitions using /etc/fstab. Use `glabel status` to receive a list of file systems and their corresponding `ufsid` labels:

```
% glabel status
                  Name  Status  Components
ufsid/486b6fc38d330916     N/A  ad4s1d
ufsid/486b6fc16926168e     N/A  ad4s1f
```

In the above example, ad4s1d represents /var, while ad4s1f represents /usr. Using the `ufsid` values shown, these partitions may now be mounted with the following entries in /etc/fstab:

```
/dev/ufsid/486b6fc38d330916        /var        ufs        rw        2      2
/dev/ufsid/486b6fc16926168e        /usr        ufs        rw        2      2
```

Any partitions with `ufsid` labels can be mounted in this way, eliminating the need to manually create permanent labels, while still enjoying the benefits of device name independent mounting.

22.8. UFS Journaling Through GEOM

Support for journals on UFS file systems is available on FreeBSD. The implementation is provided through the GEOM subsystem and is configured using `gjournal`. Unlike other file system journaling implementations, the `gjournal` method is block based and not implemented as part of the file system. It is a GEOM extension.

Journaling stores a log of file system transactions, such as changes that make up a complete disk write operation, before meta-data and file writes are committed to the disk. This transaction log can later be replayed to redo file system transactions, preventing file system inconsistencies.

This method provides another mechanism to protect against data loss and inconsistencies of the file system. Unlike Soft Updates, which tracks and enforces meta-data updates, and snapshots, which create an image of the file system, a log is stored in disk space specifically for this task. For better performance, the journal may be stored on another disk. In this configuration, the journal provider or storage device should be listed after the device to enable journaling on.

The GENERIC kernel provides support for `gjournal`. To automatically load the geom_journal.ko kernel module at boot time, add the following line to /boot/loader.conf:

```
geom_journal_load="YES"
```

If a custom kernel is used, ensure the following line is in the kernel configuration file:

```
options GEOM_JOURNAL
```

Once the module is loaded, a journal can be created on a new file system using the following steps. In this example, da4 is a new SCSI disk:

```
# gjournal load
# gjournal label /dev/da4
```

This will load the module and create a /dev/da4.journal device node on /dev/da4.

A UFS file system may now be created on the journaled device, then mounted on an existing mount point:

```
# newfs -O 2 -J /dev/da4.journal
# mount /dev/da4.journal /mnt
```

In the case of several slices, a journal will be created for each individual slice. For instance, if ad4s1 and ad4s2 are both slices, then `gjournal` will create ad4s1.journal and ad4s2.journal.

Journaling may also be enabled on current file systems by using `tunefs`. However, *always* make a backup before attempting to alter an existing file system. In most cases, `gjournal` will fail if it is unable to create the journal, but this does not protect against data loss incurred as a result of misusing `tunefs`. Refer to gjournal(8) and tunefs(8) for more information about these commands.

It is possible to journal the boot disk of a FreeBSD system. Refer to the article Implementing UFS Journaling on a Desktop PC for detailed instructions.

Chapter 23. The Z File System (ZFS)

ZFS is an advanced file system designed to solve major problems found in previous storage subsystem software.

Originally developed at Sun™, ongoing open source ZFS development has moved to the OpenZFS Project.

ZFS has three major design goals:

- Data integrity: All data includes a checksum of the data. ZFS calculates checksums and writes them along with the data. When reading that data later, ZFS recalculates the checksums. If the checksums do not match, meaning detecting one or more data errors, ZFS will attempt to automatically correct errors when ditto-, mirror-, or parity-blocks are available.
- Pooled storage: adding physical storage devices to a pool, and allocating storage space from that shared pool. Space is available to all file systems and volumes, and increases by adding new storage devices to the pool.
- Performance: caching mechanisms provide increased performance. ARC is an advanced memory-based read cache. ZFS provides a second level disk-based read cache with L2ARC, and a disk-based synchronous write cache named ZIL.

A complete list of features and terminology is in ZFS Features and Terminology.

23.1. What Makes ZFS Different

More than a file system, ZFS is fundamentally different from traditional file systems. Combining the traditionally separate roles of volume manager and file system provides ZFS with unique advantages. The file system is now aware of the underlying structure of the disks. Traditional file systems could exist on a single disk alone at a time. If there were two disks then creating two separate file systems was necessary. A traditional hardware RAID configuration avoided this problem by presenting the operating system with a single logical disk made up of the space provided by physical disks on top of which the operating system placed a file system. Even with software RAID solutions like those provided by GEOM, the UFS file system living on top of the RAID believes it’s dealing with a single device. ZFS' combination of the volume manager and the file system solves this and allows the creation of file systems that all share a pool of available storage. One big advantage of ZFS' awareness of the physical disk layout is that existing file systems grow automatically when adding extra disks to the pool. This new space then becomes available to the file systems. ZFS can also apply different properties to each file system. This makes it useful to create separate file systems and datasets instead of a single monolithic file system.

23.2. Quick Start Guide

FreeBSD can mount ZFS pools and datasets during system initialization. To enable it, run:

```
# service zfs enable
```

or add this line to /etc/rc.conf:

```
zfs_enable="YES"
```

Then start the service:

```
# service zfs start
```

The examples in this section assume three SCSI disks with the device names da0, da1, and da2. Users of SATA hardware should instead use ada device names.

23.2.1. Single Disk Pool

To create a simple, non-redundant pool using a single disk device:

```
# zpool create example /dev/da0
```

To view the new pool, review the output of `df`:

```
# df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a   2026030  235230  1628718    13%    /
devfs               1       1        0   100%    /dev
/dev/ad0s1d  54098308 1032846 48737598     2%    /usr
example      17547136       0 17547136     0%    /example
```

This output shows creating and mounting of the `example` pool, and that is now accessible as a file system. Create files for users to browse:

```
# cd /example
# ls
# touch testfile
# ls -al
total 4
drwxr-xr-x   2 root  wheel    3 Aug 29 23:15 .
drwxr-xr-x  21 root  wheel  512 Aug 29 23:12 ..
-rw-r--r--   1 root  wheel    0 Aug 29 23:15 testfile
```

This pool is not using any advanced ZFS features and properties yet. To create a dataset on this pool with compression enabled:

```
# zfs create example/compressed
# zfs set compression=gzip example/compressed
```

The `example/compressed` dataset is now a ZFS compressed file system. Try copying some large files to /example/compressed.

Disable compression with:

```
# zfs set compression=off example/compressed
```

To unmount a file system, use `zfs umount` and then verify with `df`:

```
# zfs umount example/compressed
# df
Filesystem  1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a   2026030  235232  1628716    13%    /
devfs               1       1        0   100%    /dev
/dev/ad0s1d  54098308 1032864 48737580     2%    /usr
example      17547008       0 17547008     0%    /example
```

To re-mount the file system to make it accessible again, use `zfs mount` and verify with `df`:

```
# zfs mount example/compressed
# df
Filesystem         1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a          2026030  235234  1628714    13%    /
devfs                      1       1        0   100%    /dev
/dev/ad0s1d         54098308 1032864 48737580     2%    /usr
example             17547008       0 17547008     0%    /example
example/compressed  17547008       0 17547008     0%    /example/compressed
```

Running `mount` shows the pool and file systems:

```
# mount
/dev/ad0s1a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/ad0s1d on /usr (ufs, local, soft-updates)
example on /example (zfs, local)
example/compressed on /example/compressed (zfs, local)
```

Use ZFS datasets like any file system after creation. Set other available features on a per-dataset basis when needed. The example below creates a new file system called `data`. It assumes the file system contains important files and configures it to store two copies of each data block.

```
# zfs create example/data
# zfs set copies=2 example/data
```

Use `df` to see the data and space usage:

```
# df
Filesystem         1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a          2026030  235234  1628714    13%    /
devfs                      1       1        0   100%    /dev
/dev/ad0s1d         54098308 1032864 48737580     2%    /usr
example             17547008       0 17547008     0%    /example
example/compressed  17547008       0 17547008     0%    /example/compressed
example/data        17547008       0 17547008     0%    /example/data
```

Notice that all file systems in the pool have the same available space. Using `df` in these examples shows that the file systems use the space they need and all draw from the same pool. ZFS gets rid of concepts such as volumes and partitions, and allows several file systems to share the same pool.

To destroy the file systems and then the pool that is no longer needed:

```
# zfs destroy example/compressed
# zfs destroy example/data
# zpool destroy example
```

23.2.2. RAID-Z

Disks fail. One way to avoid data loss from disk failure is to use RAID. ZFS supports this feature in its pool design. RAID-Z pools require three or more disks but provide more usable space than mirrored pools.

This example creates a RAID-Z pool, specifying the disks to add to the pool:

```
# zpool create storage raidz da0 da1 da2
```

Sun™ recommends that the number of devices used in a RAID-Z configuration be between three and nine. For environments requiring a single pool consisting of 10 disks or more, consider breaking it up into smaller RAID-Z groups. If two disks are available, ZFS mirroring provides redundancy if required. Refer to zpool(8) for more details.

The previous example created the `storage` zpool. This example makes a new file system called `home` in that pool:

```
# zfs create storage/home
```

Enable compression and store an extra copy of directories and files:

```
# zfs set copies=2 storage/home
# zfs set compression=gzip storage/home
```

To make this the new home directory for users, copy the user data to this directory and create the appropriate symbolic links:

```
# cp -rp /home/* /storage/home
# rm -rf /home /usr/home
# ln -s /storage/home /home
# ln -s /storage/home /usr/home
```

Users data is now stored on the freshly-created /storage/home. Test by adding a new user and logging in as that user.

Create a file system snapshot to roll back to later:

```
# zfs snapshot storage/home@08-30-08
```

ZFS creates snapshots of a dataset, not a single directory or file.

The `@` character is a delimiter between the file system name or the volume name. Before deleting an important directory, back up the file system, then roll back to an earlier snapshot in which the directory still exists:

```
# zfs rollback storage/home@08-30-08
```

To list all available snapshots, run `ls` in the file system's .zfs/snapshot directory. For example, to see the snapshot taken:

```
# ls /storage/home/.zfs/snapshot
```

Write a script to take regular snapshots of user data. Over time, snapshots can use up a lot of disk space. Remove the previous snapshot using the command:

```
# zfs destroy storage/home@08-30-08
```

After testing, make /storage/home the real /home with this command:

```
# zfs set mountpoint=/home storage/home
```

Run `df` and `mount` to confirm that the system now treats the file system as the real /home:

```
# mount
/dev/ad0s1a on / (ufs, local)
devfs on /dev (devfs, local)
/dev/ad0s1d on /usr (ufs, local, soft-updates)
```

```
storage on /storage (zfs, local)
storage/home on /home (zfs, local)
# df
Filesystem   1K-blocks    Used    Avail Capacity  Mounted on
/dev/ad0s1a    2026030  235240  1628708    13%    /
devfs                1       1        0   100%    /dev
/dev/ad0s1d   54098308 1032826 48737618     2%    /usr
storage       26320512       0 26320512     0%    /storage
storage/home  26320512       0 26320512     0%    /home
```

This completes the RAID-Z configuration. Add daily status updates about the created file systems to the nightly periodic(8) runs by adding this line to /etc/periodic.conf:

```
daily_status_zfs_enable="YES"
```

23.2.3. Recovering RAID-Z

Every software RAID has a method of monitoring its `state`. View the status of RAID-Z devices using:

```
# zpool status -x
```

If all pools are Online and everything is normal, the message shows:

```
all pools are healthy
```

If there is a problem, perhaps a disk being in the Offline state, the pool state will look like this:

```
  pool: storage
 state: DEGRADED
status: One or more devices has been taken offline by the administrator.
    Sufficient replicas exist for the pool to continue functioning in a
    degraded state.
action: Online the device using 'zpool online' or replace the device with
    'zpool replace'.
 scrub: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    storage     DEGRADED     0     0     0
      raidz1    DEGRADED     0     0     0
        da0     ONLINE       0     0     0
        da1     OFFLINE      0     0     0
        da2     ONLINE       0     0     0

errors: No known data errors
```

"OFFLINE" shows the administrator took da1 offline using:

```
# zpool offline storage da1
```

Power down the computer now and replace da1. Power up the computer and return da1 to the pool:

```
# zpool replace storage da1
```

Next, check the status again, this time without `-x` to display all pools:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: resilver completed with 0 errors on Sat Aug 30 19:44:11 2008
config:

    NAME        STATE     READ WRITE CKSUM
    storage     ONLINE       0     0     0
      raidz1    ONLINE       0     0     0
        da0     ONLINE       0     0     0
        da1     ONLINE       0     0     0
        da2     ONLINE       0     0     0

errors: No known data errors
```

In this example, everything is normal.

23.2.4. Data Verification

ZFS uses checksums to verify the integrity of stored data. Creating file systems automatically enables them.

> Disabling Checksums is possible but *not* recommended! Checksums take little storage space and provide data integrity. Most ZFS features will not work properly with checksums disabled. Disabling these checksums will not increase performance noticeably.

Verifying the data checksums (called *scrubbing*) ensures integrity of the `storage` pool with:

```
# zpool scrub storage
```

The duration of a scrub depends on the amount of data stored. Larger amounts of data will take proportionally longer to verify. Since scrubbing is I/O intensive, ZFS allows a single scrub to run at a time. After scrubbing completes, view the status with `zpool status`:

```
# zpool status storage
 pool: storage
 state: ONLINE
 scrub: scrub completed with 0 errors on Sat Jan 26 19:57:37 2013
config:

    NAME        STATE     READ WRITE CKSUM
    storage     ONLINE       0     0     0
      raidz1    ONLINE       0     0     0
        da0     ONLINE       0     0     0
        da1     ONLINE       0     0     0
        da2     ONLINE       0     0     0

errors: No known data errors
```

Displaying the completion date of the last scrubbing helps decide when to start another. Routine scrubs help protect data from silent corruption and ensure the integrity of the pool.

Refer to zfs(8) and zpool(8) for other ZFS options.

23.3. `zpool` Administration

ZFS administration uses two main utilities. The `zpool` utility controls the operation of the pool and allows adding, removing, replacing, and managing disks. The `zfs` utility allows creating, destroying, and managing datasets, both file systems and volumes.

23.3.1. Creating and Destroying Storage Pools

Creating a ZFS storage pool requires permanent decisions, as the pool structure cannot change after creation. The most important decision is which types of vdevs to group the physical disks into. See the list of vdev types for details about the possible options. After creating the pool, most vdev types do not allow adding disks to the vdev. The exceptions are mirrors, which allow adding new disks to the vdev, and stripes, which upgrade to mirrors by attaching a new disk to the vdev. Although adding new vdevs expands a pool, the pool layout cannot change after pool creation. Instead, back up the data, destroy the pool, and recreate it.

Create a simple mirror pool:

```
# zpool create mypool mirror /dev/ada1 /dev/ada2
# zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
```

```
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0     0

errors: No known data errors
```

To create more than one vdev with a single command, specify groups of disks separated by the vdev type keyword, `mirror` in this example:

```
# zpool create mypool mirror /dev/ada1 /dev/ada2 mirror /dev/ada3 /dev/ada4
# zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada1    ONLINE       0     0     0
            ada2    ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            ada3    ONLINE       0     0     0
            ada4    ONLINE       0     0     0

errors: No known data errors
```

Pools can also use partitions rather than whole disks. Putting ZFS in a separate partition allows the same disk to have other partitions for other purposes. In particular, it allows adding partitions with bootcode and file systems needed for booting. This allows booting from disks that are also members of a pool. ZFS adds no performance penalty on FreeBSD when using a partition rather than a whole disk. Using partitions also allows the administrator to *under-provision* the disks, using less than the full capacity. If a future replacement disk of the same nominal size as the original actually has a slightly smaller capacity, the smaller partition will still fit, using the replacement disk.

Create a RAID-Z2 pool using partitions:

```
# zpool create mypool raidz2 /dev/ada0p3 /dev/ada1p3 /dev/ada2p3 /dev/ada3p3
/dev/ada4p3 /dev/ada5p3
# zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          raidz2-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0
            ada3p3  ONLINE       0     0     0
            ada4p3  ONLINE       0     0     0
            ada5p3  ONLINE       0     0     0

errors: No known data errors
```

Destroy a pool that is no longer needed to reuse the disks. Destroying a pool requires unmounting the file systems in that pool first. If any dataset is in use, the unmount operation fails without destroying the pool. Force the pool destruction with `-f`. This can cause undefined behavior in applications which had open files on those datasets.

23.3.2. Adding and Removing Devices

Two ways exist for adding disks to a pool: attaching a disk to an existing vdev with `zpool attach`, or adding vdevs to the pool with `zpool add`. Some vdev types allow adding disks to the vdev after creation.

A pool created with a single disk lacks redundancy. It can detect corruption but can not repair it, because there is no other copy of the data. The copies property may be able to recover from a small failure such as a bad sector, but does not provide the same level of protection as mirroring or RAID-Z. Starting with a pool consisting of a single disk vdev, use `zpool attach` to add a new disk to the vdev, creating a mirror. Also use `zpool attach` to add new disks to a mirror group, increasing redundancy and read performance. When partitioning the disks used for the pool, replicate the layout of the first disk on to the second. Use `gpart backup` and `gpart restore` to make this process easier.

Upgrade the single disk (stripe) vdev ada0p3 to a mirror by attaching ada1p3:

```
# zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          ada0p3    ONLINE       0     0     0

errors: No known data errors
# zpool attach mypool ada0p3 ada1p3
Make sure to wait until resilvering finishes before rebooting.

If you boot from pool 'mypool', you may need to update boot code on newly attached
disk _ada1p3_.

Assuming you use GPT partitioning and _da0_ is your new boot disk you may use the
following command:
        gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 da0
# gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada1
bootcode written to ada1
# zpool status
  pool: mypool
 state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
  scan: resilver in progress since Fri May 30 08:19:19 2014
        527M scanned out of 781M at 47.9M/s, 0h0m to go
        527M resilvered, 67.53% done
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0  (resilvering)

errors: No known data errors
# zpool status
  pool: mypool
 state: ONLINE
  scan: resilvered 781M in 0h0m with 0 errors on Fri May 30 08:15:58 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0

errors: No known data errors
```

When adding disks to the existing vdev is not an option, as for RAID-Z, an alternative method is to add another vdev to the pool. Adding vdevs provides higher performance by distributing writes across the vdevs. Each vdev provides its own redundancy. Mixing vdev types like `mirror` and `RAID-Z` is possible but discouraged. Adding a non-redundant vdev to a pool containing mirror or RAID-Z vdevs risks the data on the entire pool. Distributing writes means a failure of the non-redundant disk will result in the loss of a fraction of every block written to the pool.

ZFS stripes data across each of the vdevs. For example, with two mirror vdevs, this is effectively a RAID 10 that stripes writes across two sets of mirrors. ZFS allocates space so that each vdev reaches 100% full at the same time. Having vdevs with different amounts of free space will lower performance, as more data writes go to the less full vdev.

When attaching new devices to a boot pool, remember to update the bootcode.

Attach a second mirror group (ada2p3 and ada3p3) to the existing mirror:

```
# zpool status
  pool: mypool
 state: ONLINE
  scan: resilvered 781M in 0h0m with 0 errors on Fri May 30 08:19:35 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0

errors: No known data errors
# zpool add mypool mirror ada2p3 ada3p3
# gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada2
bootcode written to ada2
# gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada3
bootcode written to ada3
# zpool status
  pool: mypool
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Fri May 30 08:29:51 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0
          mirror-1  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0
            ada3p3  ONLINE       0     0     0

errors: No known data errors
```

Removing vdevs from a pool is impossible and removal of disks from a mirror is exclusive if there is enough remaining redundancy. If a single disk remains in a mirror group, that group ceases to be a mirror and becomes a stripe, risking the entire pool if that remaining disk fails.

Remove a disk from a three-way mirror group:

```
# zpool status
  pool: mypool
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Fri May 30 08:29:51 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0

errors: No known data errors
# zpool detach mypool ada2p3
# zpool status
  pool: mypool
 state: ONLINE
  scan: scrub repaired 0 in 0h0m with 0 errors on Fri May 30 08:29:51 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0

errors: No known data errors
```

23.3.3. Checking the Status of a Pool

Pool status is important. If a drive goes offline or ZFS detects a read, write, or checksum error, the corresponding error count increases. The `status` output shows the configuration and status of each device in the pool and the status of the entire pool. Actions to take and details about the last `scrub` are also shown.

```
# zpool status
  pool: mypool
 state: ONLINE
  scan: scrub repaired 0 in 2h25m with 0 errors on Sat Sep 14 04:25:50 2013
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          raidz2-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0
            ada3p3  ONLINE       0     0     0
            ada4p3  ONLINE       0     0     0
            ada5p3  ONLINE       0     0     0

errors: No known data errors
```

23.3.4. Clearing Errors

When detecting an error, ZFS increases the read, write, or checksum error counts. Clear the error message and reset the counts with `zpool clear mypool`. Clearing the error state can be important for automated scripts that alert the administrator when the pool encounters an error. Without clearing old errors, the scripts may fail to report further errors.

23.3.5. Replacing a Functioning Device

It may be desirable to replace one disk with a different disk. When replacing a working disk, the process keeps the old disk online during the replacement. The pool never enters a degraded state, reducing the risk of data loss. Running `zpool replace` copies the data from the old disk to the new one. After the operation completes, ZFS disconnects the old disk from the vdev. If the new disk is larger than the old disk, it may be possible to grow the zpool, using the new space. See Growing a Pool.

Replace a functioning device in the pool:

```
# zpool status
  pool: mypool
 state: ONLINE
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0

errors: No known data errors
# zpool replace mypool ada1p3 ada2p3
Make sure to wait until resilvering finishes before rebooting.

When booting from the pool 'zroot', update the boot code on the newly attached disk
'ada2p3'.

Assuming GPT partitioning is used and [.filename]#da0# is the new boot disk, use the
following command:

        gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 da0
# gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada2
# zpool status
  pool: mypool
 state: ONLINE
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
  scan: resilver in progress since Mon Jun  2 14:21:35 2014
        604M scanned out of 781M at 46.5M/s, 0h0m to go
        604M resilvered, 77.39% done
config:

        NAME                STATE     READ WRITE CKSUM
        mypool              ONLINE       0     0     0
          mirror-0          ONLINE       0     0     0
            ada0p3          ONLINE       0     0     0
            replacing-1     ONLINE       0     0     0
              ada1p3        ONLINE       0     0     0
              ada2p3        ONLINE       0     0     0  (resilvering)

errors: No known data errors
# zpool status
  pool: mypool
 state: ONLINE
  scan: resilvered 781M in 0h0m with 0 errors on Mon Jun  2 14:21:52 2014
config:

        NAME          STATE     READ WRITE CKSUM
        mypool        ONLINE       0     0     0
          mirror-0    ONLINE       0     0     0
            ada0p3    ONLINE       0     0     0
            ada2p3    ONLINE       0     0     0

errors: No known data errors
```

23.3.6. Dealing with Failed Devices

When a disk in a pool fails, the vdev to which the disk belongs enters the degraded state. The data is still available, but with reduced performance because ZFS computes missing data from the available redundancy. To restore the vdev to a fully functional state, replace the failed physical device. ZFS is then instructed to begin the resilver operation. ZFS recomputes data on the failed device from available redundancy and writes it to the replacement device. After completion, the vdev returns to online status.

If the vdev does not have any redundancy, or if devices have failed and there is not enough redundancy to compensate, the pool enters the faulted state. Unless enough devices can reconnect the pool becomes inoperative requiring a data restore from backups.

When replacing a failed disk, the name of the failed disk changes to the GUID of the new disk. A new device name parameter for `zpool replace` is not required if the replacement device has the same device name.

Replace a failed disk using `zpool replace`:

```
# zpool status
  pool: mypool
 state: DEGRADED
status: One or more devices could not be opened.  Sufficient replicas exist for
```

```
        the pool to continue functioning in a degraded state.
action: Attach the missing device and online it using 'zpool online'.
   see: http://illumos.org/msg/ZFS-8000-2Q
  scan: none requested
config:

        NAME                    STATE     READ WRITE CKSUM
        mypool                  DEGRADED     0     0     0
          mirror-0              DEGRADED     0     0     0
            ada0p3              ONLINE       0     0     0
            316502962686821739  UNAVAIL      0     0     0  was /dev/ada1p3

errors: No known data errors
# zpool replace mypool 316502962686821739 ada2p3
# zpool status
  pool: mypool
 state: DEGRADED
status: One or more devices is currently being resilvered.  The pool will
        continue to function, possibly in a degraded state.
action: Wait for the resilver to complete.
  scan: resilver in progress since Mon Jun  2 14:52:21 2014
        641M scanned out of 781M at 49.3M/s, 0h0m to go
        640M resilvered, 82.04% done
config:

        NAME                        STATE     READ WRITE CKSUM
        mypool                      DEGRADED     0     0     0
          mirror-0                  DEGRADED     0     0     0
            ada0p3                  ONLINE       0     0     0
            replacing-1             UNAVAIL      0     0     0
              15732067398082357289  UNAVAIL      0     0     0  was /dev/ada1p3/old
              ada2p3                ONLINE       0     0     0  (resilvering)

errors: No known data errors
# zpool status
  pool: mypool
 state: ONLINE
  scan: resilvered 781M in 0h0m with 0 errors on Mon Jun  2 14:52:38 2014
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0

errors: No known data errors
```

23.3.7. Scrubbing a Pool

Routinely scrub pools, ideally at least once every month. The `scrub` operation is disk-intensive and will reduce performance while running. Avoid high-demand periods when scheduling `scrub` or use `vfs.zfs.scrub_delay` to adjust the relative priority of the `scrub` to keep it from slowing down other workloads.

```
# zpool scrub mypool
# zpool status
  pool: mypool
 state: ONLINE
  scan: scrub in progress since Wed Feb 19 20:52:54 2014
        116G scanned out of 8.60T at 649M/s, 3h48m to go
        0 repaired, 1.32% done
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          raidz2-0  ONLINE       0     0     0
            ada0p3  ONLINE       0     0     0
            ada1p3  ONLINE       0     0     0
            ada2p3  ONLINE       0     0     0
            ada3p3  ONLINE       0     0     0
            ada4p3  ONLINE       0     0     0
            ada5p3  ONLINE       0     0     0

errors: No known data errors
```

To cancel a scrub operation if needed, run `zpool scrub -s mypool`.

23.3.8. Self-Healing

The checksums stored with data blocks enable the file system to *self-heal*. This feature will automatically repair data whose checksum does not match the one recorded on another device that is part of the storage pool. For example, a mirror configuration with two disks where one drive is starting to malfunction and cannot properly store the data any more. This is worse when the data was not accessed for a long time, as with long term archive storage. Traditional file systems need to run commands that check and repair the data like fsck(8). These commands take time, and in severe cases, an administrator has to decide which repair operation to perform. When ZFS detects a data block with a mismatched checksum, it tries to read the data from the mirror disk. If that disk can provide the correct data, ZFS will give that to the application and correct the data on the disk with the wrong checksum. This happens without any interaction from a system administrator during normal pool operation.

The next example shows this self-healing behavior by creating a mirrored pool of disks /dev/ada0 and /dev/ada1.

```
# zpool create healer mirror /dev/ada0 /dev/ada1
# zpool status healer
  pool: healer
 state: ONLINE
  scan: none requested
config:

    NAME        STATE     READ WRITE CKSUM
    healer      ONLINE       0     0     0
      mirror-0  ONLINE       0     0     0
       ada0     ONLINE       0     0     0
       ada1     ONLINE       0     0     0

errors: No known data errors
# zpool list
NAME     SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
healer   960M  92.5K   960M         -         -     0%     0%  1.00x  ONLINE  -
```

Copy some important data to the pool to protect from data errors using the self-healing feature and create a checksum of the pool for later comparison.

```
# cp /some/important/data /healer
# zfs list
NAME     SIZE  ALLOC   FREE    CAP  DEDUP  HEALTH  ALTROOT
healer   960M  67.7M   892M     7%  1.00x  ONLINE  -
# sha1 /healer > checksum.txt
# cat checksum.txt
SHA1 (/healer) = 2753eff56d77d9a536ece6694bf0a82740344d1f
```

Simulate data corruption by writing random data to the beginning of one of the disks in the mirror. To keep ZFS from healing the data when detected, export the pool before the corruption and import it again afterwards.

This is a dangerous operation that can destroy vital data, shown here for demonstration alone. **Do not try** it during normal operation of a storage pool. Nor should this intentional corruption example run on any disk with a file system not using ZFS on another partition in it. Do not use any other disk device names other than the ones that are part of the pool. Ensure proper backups of the pool exist and test them before running the command!

```
# zpool export healer
# dd if=/dev/random of=/dev/ada1 bs=1m count=200
200+0 records in
200+0 records out
209715200 bytes transferred in 62.992162 secs (3329227 bytes/sec)
# zpool import healer
```

The pool status shows that one device has experienced an error. Note that applications reading data from the pool did not receive any incorrect data. ZFS provided data from the ada0 device with the correct checksums. To find the device with the wrong checksum, look for one whose `CKSUM` column contains a nonzero value.

```
# zpool status healer
    pool: healer
   state: ONLINE
  status: One or more devices has experienced an unrecoverable error.  An
          attempt was made to correct the error.  Applications are unaffected.
  action: Determine if the device needs to be replaced, and clear the errors
          using 'zpool clear' or replace the device with 'zpool replace'.
     see: http://illumos.org/msg/ZFS-8000-4J
    scan: none requested
  config:

      NAME        STATE     READ WRITE CKSUM
      healer      ONLINE       0     0     0
        mirror-0  ONLINE       0     0     0
         ada0     ONLINE       0     0     0
         ada1     ONLINE       0     0     1

errors: No known data errors
```

ZFS detected the error and handled it by using the redundancy present in the unaffected ada0 mirror disk. A checksum comparison with the original one will reveal whether the pool is consistent again.

```
# sha1 /healer >> checksum.txt
# cat checksum.txt
SHA1 (/healer) = 2753eff56d77d9a536ece6694bf0a82740344d1f
SHA1 (/healer) = 2753eff56d77d9a536ece6694bf0a82740344d1f
```

Generate checksums before and after the intentional tampering while the pool data still matches. This shows how ZFS is capable of detecting and correcting any errors automatically when the checksums differ. Note this is possible with enough redundancy present in the pool. A pool consisting of a single device has no self-healing capabilities. That is also the reason why checksums are so important in ZFS; do not disable them for any reason. ZFS requires no fsck(8) or similar file system consistency check program to detect and correct this, and keeps the pool available while there is a problem. A scrub operation is now required to overwrite the corrupted data on ada1.

```
# zpool scrub healer
# zpool status healer
  pool: healer
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
            attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
```

```
            using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://illumos.org/msg/ZFS-8000-4J
  scan: scrub in progress since Mon Dec 10 12:23:30 2012
        10.4M scanned out of 67.0M at 267K/s, 0h3m to go
        9.63M repaired, 15.56% done
config:

    NAME        STATE     READ WRITE CKSUM
    healer      ONLINE       0     0     0
      mirror-0  ONLINE       0     0     0
       ada0     ONLINE       0     0     0
       ada1     ONLINE       0     0   627  (repairing)

errors: No known data errors
```

The scrub operation reads data from ada0 and rewrites any data with a wrong checksum on ada1, shown by the `(repairing)` output from `zpool status`. After the operation is complete, the pool status changes to:

```
# zpool status healer
  pool: healer
 state: ONLINE
status: One or more devices has experienced an unrecoverable error.  An
        attempt was made to correct the error.  Applications are unaffected.
action: Determine if the device needs to be replaced, and clear the errors
             using 'zpool clear' or replace the device with 'zpool replace'.
   see: http://illumos.org/msg/ZFS-8000-4J
  scan: scrub repaired 66.5M in 0h2m with 0 errors on Mon Dec 10 12:26:25 2012
config:

    NAME        STATE     READ WRITE CKSUM
    healer      ONLINE       0     0     0
      mirror-0  ONLINE       0     0     0
       ada0     ONLINE       0     0     0
       ada1     ONLINE       0     0 2.72K

errors: No known data errors
```

After the scrubbing operation completes with all the data synchronized from ada0 to ada1, clear the error messages from the pool status by running `zpool clear`.

```
# zpool clear healer
# zpool status healer
  pool: healer
 state: ONLINE
  scan: scrub repaired 66.5M in 0h2m with 0 errors on Mon Dec 10 12:26:25 2012
config:
     NAME        STATE     READ WRITE CKSUM
     healer      ONLINE       0     0     0
       mirror-0  ONLINE       0     0     0
        ada0     ONLINE       0     0     0
        ada1     ONLINE       0     0     0

errors: No known data errors
```

The pool is now back to a fully working state, with all error counts now zero.

23.3.9. Growing a Pool

The smallest device in each vdev limits the usable size of a redundant pool. Replace the smallest device with a larger device. After completing a replace or resilver operation, the pool can grow to use the capacity of the new device. For example, consider a mirror of a 1 TB drive and a 2 TB drive. The usable space is 1 TB. When replacing the 1 TB drive with another 2 TB drive, the resilvering process copies the existing data onto the new drive. As both of the devices now have 2 TB capacity, the mirror's available space grows to 2 TB.

Start expansion by using `zpool online -e` on each device. After expanding all devices, the extra space becomes available to the pool.

23.3.10. Importing and Exporting Pools

Export pools before moving them to another system. ZFS unmounts all datasets, marking each device as exported but still locked to prevent use by other disks. This allows pools to be *imported* on other machines, other operating systems that support ZFS, and even different hardware architectures (with some caveats, see zpool(8)). When a dataset has open files, use `zpool export -f` to force exporting the pool. Use this with caution. The datasets are forcibly unmounted, potentially resulting in unexpected behavior by the applications which had open files on those datasets.

Export a pool that is not in use:

```
# zpool export mypool
```

Importing a pool automatically mounts the datasets. If this is undesired behavior, use `zpool import -N` to prevent it. `zpool import -o` sets temporary properties for this specific import. `zpool import altroot=` allows importing a pool with a base mount point instead of the root of the file system. If the pool was last used on a different system and was not properly exported, force the import using `zpool import -f`. `zpool import -a` imports all pools that do not appear to be in use by another system.

List all available pools for import:

```
# zpool import
   pool: mypool
     id: 9930174748043525076
  state: ONLINE
```

```
 action: The pool can be imported using its name or numeric identifier.
 config:

        mypool      ONLINE
          ada2p3    ONLINE
```

Import the pool with an alternative root directory:

```
# zpool import -o altroot=/mnt mypool
# zfs list
zfs list
NAME                 USED  AVAIL  REFER  MOUNTPOINT
mypool               110K  47.0G    31K  /mnt/mypool
```

23.3.11. Upgrading a Storage Pool

After upgrading FreeBSD, or if importing a pool from a system using an older version, manually upgrade the pool to the latest ZFS version to support newer features. Consider whether the pool may ever need importing on an older system before upgrading. Upgrading is a one-way process. Upgrade older pools is possible, but downgrading pools with newer features is not.

Upgrade a v28 pool to support `Feature Flags`:

```
# zpool status
  pool: mypool
 state: ONLINE
status: The pool is formatted using a legacy on-disk format.  The pool can
        still be used, but some features are unavailable.
action: Upgrade the pool using 'zpool upgrade'.  Once this is done, the
        pool will no longer be accessible on software that does not support feat
        flags.
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
        ada0    ONLINE       0     0     0
        ada1    ONLINE       0     0     0

errors: No known data errors
# zpool upgrade
This system supports ZFS pool feature flags.

The following pools are formatted with legacy version numbers and are upgraded to use
feature flags.
After being upgraded, these pools will no longer be accessible by software that does
not support feature flags.
```

```
VER  POOL
---  ------------
28   mypool

Use 'zpool upgrade -v' for a list of available legacy versions.
Every feature flags pool has all supported features enabled.
# zpool upgrade mypool
This system supports ZFS pool feature flags.

Successfully upgraded 'mypool' from version 28 to feature flags.
Enabled the following features on 'mypool':
  async_destroy
  empty_bpobj
  lz4_compress
  multi_vdev_crash_dump
```

The newer features of ZFS will not be available until `zpool upgrade` has completed. Use `zpool upgrade -v` to see what new features the upgrade provides, as well as which features are already supported.

Upgrade a pool to support new feature flags:

```
# zpool status
  pool: mypool
 state: ONLINE
status: Some supported features are not enabled on the pool. The pool can
        still be used, but some features are unavailable.
action: Enable all features using 'zpool upgrade'. Once this is done,
        the pool may no longer be accessible by software that does not support
        the features. See zpool-features(7) for details.
  scan: none requested
config:

        NAME        STATE     READ WRITE CKSUM
        mypool      ONLINE       0     0     0
          mirror-0  ONLINE       0     0     0
        ada0    ONLINE       0     0     0
        ada1    ONLINE       0     0     0

errors: No known data errors
# zpool upgrade
This system supports ZFS pool feature flags.

All pools are formatted using feature flags.

Some supported features are not enabled on the following pools. Once a
feature is enabled the pool may become incompatible with software
that does not support the feature. See zpool-features(7) for details.
```

```
POOL  FEATURE
---------------
zstore
      multi_vdev_crash_dump
      spacemap_histogram
      enabled_txg
      hole_birth
      extensible_dataset
      bookmarks
      filesystem_limits
# zpool upgrade mypool
This system supports ZFS pool feature flags.

Enabled the following features on 'mypool':
  spacemap_histogram
  enabled_txg
  hole_birth
  extensible_dataset
  bookmarks
  filesystem_limits
```

Update the boot code on systems that boot from a pool to support the new pool version. Use `gpart bootcode` on the partition that contains the boot code. Two types of bootcode are available, depending on way the system boots: GPT (the most common option) and EFI (for more modern systems).

For legacy boot using GPT, use the following command:

```
# gpart bootcode -b /boot/pmbr -p /boot/gptzfsboot -i 1 ada1
```

For systems using EFI to boot, execute the following command:

```
# mount -t msdosfs /dev/ada1p1 /boot/efi
# cp /boot/loader.efi /boot/efi/efi/boot/BOOTX64.efi
# cp /boot/loader.efi /boot/efi/efi/freebsd/loader.efi
```

These instructions apply to amd64 systems. Depending on the architecture, the file has a different name. Refer to uefi(8) to find the names of the architecture-specific files.

Apply the bootcode to all bootable disks in the pool. See gpart(8) for more information.

23.3.12. Displaying Recorded Pool History

ZFS records commands that change the pool, including creating datasets, changing properties, or replacing a disk. Reviewing history about a pool's creation is useful, as is checking which user performed a specific action and when. History is not kept in a log file, but is part of the pool itself. The command to review this history is aptly named `zpool history`:

```
# zpool history
History for 'tank':
2013-02-26.23:02:35 zpool create tank mirror /dev/ada0 /dev/ada1
2013-02-27.18:50:58 zfs set atime=off tank
2013-02-27.18:51:09 zfs set checksum=fletcher4 tank
2013-02-27.18:51:18 zfs create tank/backup
```

The output shows `zpool` and `zfs` commands altering the pool in some way along with a timestamp. Commands like `zfs list` are not included. When specifying no pool name, ZFS displays history of all pools.

`zpool history` can show even more information when providing the options `-i` or `-l`. `-i` displays user-initiated events as well as internally logged ZFS events.

```
# zpool history -i
History for 'tank':
2013-02-26.23:02:35 [internal pool create txg:5] pool spa 28; zfs spa 28; zpl 5;uts
9.1-RELEASE 901000 amd64
2013-02-27.18:50:53 [internal property set txg:50] atime=0 dataset = 21
2013-02-27.18:50:58 zfs set atime=off tank
2013-02-27.18:51:04 [internal property set txg:53] checksum=7 dataset = 21
2013-02-27.18:51:09 zfs set checksum=fletcher4 tank
2013-02-27.18:51:13 [internal create txg:55] dataset = 39
2013-02-27.18:51:18 zfs create tank/backup
```

Show more details by adding `-l`. Showing history records in a long format, including information like the name of the user who issued the command and the hostname on which the change happened.

```
# zpool history -l
History for 'tank':
2013-02-26.23:02:35 zpool create tank mirror /dev/ada0 /dev/ada1 [user 0 (root) on
:global]
2013-02-27.18:50:58 zfs set atime=off tank [user 0 (root) on myzfsbox:global]
2013-02-27.18:51:09 zfs set checksum=fletcher4 tank [user 0 (root) on myzfsbox:global]
2013-02-27.18:51:18 zfs create tank/backup [user 0 (root) on myzfsbox:global]
```

The output shows that the `root` user created the mirrored pool with disks /dev/ada0 and /dev/ada1. The hostname `myzfsbox` is also shown in the commands after the pool's creation. The hostname display becomes important when exporting the pool from one system and importing on another. It's possible to distinguish the commands issued on the other system by the hostname recorded for each command.

Combine both options to `zpool history` to give the most detailed information possible for any given pool. Pool history provides valuable information when tracking down the actions performed or when needing more detailed output for debugging.

23.3.13. Performance Monitoring

A built-in monitoring system can display pool I/O statistics in real time. It shows the amount of free and used space on the pool, read and write operations performed per second, and I/O bandwidth used. By default, ZFS monitors and displays all pools in the system. Provide a pool name to limit monitoring to that pool. A basic example:

```
# zpool iostat
               capacity     operations    bandwidth
pool        alloc   free   read  write   read  write
----------  -----  -----  -----  -----  -----  -----
data         288G  1.53T      2     11  11.3K  57.1K
```

To continuously see I/O activity, specify a number as the last parameter, indicating an interval in seconds to wait between updates. The next statistic line prints after each interval. Press `Ctrl`+`C` to stop this continuous monitoring. Give a second number on the command line after the interval to specify the total number of statistics to display.

Display even more detailed I/O statistics with `-v`. Each device in the pool appears with a statistics line. This is useful for seeing read and write operations performed on each device, and can help determine if any individual device is slowing down the pool. This example shows a mirrored pool with two devices:

```
# zpool iostat -v
                            capacity     operations    bandwidth
pool                     alloc   free   read  write   read  write
-----------------------  -----  -----  -----  -----  -----  -----
data                      288G  1.53T      2     12  9.23K  61.5K
  mirror                  288G  1.53T      2     12  9.23K  61.5K
    ada1                     -      -      0      4  5.61K  61.7K
    ada2                     -      -      1      4  5.04K  61.7K
-----------------------  -----  -----  -----  -----  -----  -----
```

23.3.14. Splitting a Storage Pool

ZFS can split a pool consisting of one or more mirror vdevs into two pools. Unless otherwise specified, ZFS detaches the last member of each mirror and creates a new pool containing the same data. Be sure to make a dry run of the operation with `-n` first. This displays the details of the requested operation without actually performing it. This helps confirm that the operation will do what the user intends.

23.4. `zfs` Administration

The `zfs` utility can create, destroy, and manage all existing ZFS datasets within a pool. To manage the pool itself, use `zpool`.

23.4.1. Creating and Destroying Datasets

Unlike traditional disks and volume managers, space in ZFS is *not* preallocated. With traditional file systems, after partitioning and assigning the space, there is no way to add a new file system without adding a new disk. With ZFS, creating new file systems is possible at any time. Each *dataset* has properties including features like compression, deduplication, caching, and quotas, as well as other useful properties like readonly, case sensitivity, network file sharing, and a mount point. Nesting datasets within each other is possible and child datasets will inherit properties from their ancestors. Delegate, replicate, snapshot, jail allows administering and destroying each dataset as a unit. Creating a separate dataset for each different type or set of files has advantages. The drawbacks to having a large number of datasets are that some commands like `zfs list` will be slower, and that mounting of hundreds or even thousands of datasets will slow the FreeBSD boot process.

Create a new dataset and enable LZ4 compression on it:

```
# zfs list
NAME                  USED  AVAIL  REFER  MOUNTPOINT
mypool                781M  93.2G   144K  none
mypool/ROOT           777M  93.2G   144K  none
mypool/ROOT/default   777M  93.2G   777M  /
mypool/tmp            176K  93.2G   176K  /tmp
mypool/usr            616K  93.2G   144K  /usr
mypool/usr/home       184K  93.2G   184K  /usr/home
mypool/usr/ports      144K  93.2G   144K  /usr/ports
mypool/usr/src        144K  93.2G   144K  /usr/src
mypool/var           1.20M  93.2G   608K  /var
mypool/var/crash      148K  93.2G   148K  /var/crash
mypool/var/log        178K  93.2G   178K  /var/log
mypool/var/mail       144K  93.2G   144K  /var/mail
mypool/var/tmp        152K  93.2G   152K  /var/tmp
# zfs create -o compress=lz4 mypool/usr/mydataset
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
mypool                 781M  93.2G   144K  none
mypool/ROOT            777M  93.2G   144K  none
mypool/ROOT/default    777M  93.2G   777M  /
mypool/tmp             176K  93.2G   176K  /tmp
mypool/usr             704K  93.2G   144K  /usr
mypool/usr/home        184K  93.2G   184K  /usr/home
mypool/usr/mydataset  87.5K  93.2G  87.5K  /usr/mydataset
mypool/usr/ports       144K  93.2G   144K  /usr/ports
mypool/usr/src         144K  93.2G   144K  /usr/src
mypool/var            1.20M  93.2G   610K  /var
```

```
mypool/var/crash       148K  93.2G   148K  /var/crash
mypool/var/log         178K  93.2G   178K  /var/log
mypool/var/mail        144K  93.2G   144K  /var/mail
mypool/var/tmp         152K  93.2G   152K  /var/tmp
```

Destroying a dataset is much quicker than deleting the files on the dataset, as it does not involve scanning the files and updating the corresponding metadata.

Destroy the created dataset:

```
# zfs list
NAME                   USED  AVAIL  REFER  MOUNTPOINT
mypool                 880M  93.1G   144K  none
mypool/ROOT            777M  93.1G   144K  none
mypool/ROOT/default    777M  93.1G   777M  /
mypool/tmp             176K  93.1G   176K  /tmp
mypool/usr             101M  93.1G   144K  /usr
mypool/usr/home        184K  93.1G   184K  /usr/home
mypool/usr/mydataset   100M  93.1G   100M  /usr/mydataset
mypool/usr/ports       144K  93.1G   144K  /usr/ports
mypool/usr/src         144K  93.1G   144K  /usr/src
mypool/var            1.20M  93.1G   610K  /var
mypool/var/crash       148K  93.1G   148K  /var/crash
mypool/var/log         178K  93.1G   178K  /var/log
mypool/var/mail        144K  93.1G   144K  /var/mail
mypool/var/tmp         152K  93.1G   152K  /var/tmp
# zfs destroy mypool/usr/mydataset
# zfs list
NAME                  USED  AVAIL  REFER  MOUNTPOINT
mypool                781M  93.2G   144K  none
mypool/ROOT           777M  93.2G   144K  none
mypool/ROOT/default   777M  93.2G   777M  /
mypool/tmp            176K  93.2G   176K  /tmp
mypool/usr            616K  93.2G   144K  /usr
mypool/usr/home       184K  93.2G   184K  /usr/home
mypool/usr/ports      144K  93.2G   144K  /usr/ports
mypool/usr/src        144K  93.2G   144K  /usr/src
mypool/var           1.21M  93.2G   612K  /var
mypool/var/crash      148K  93.2G   148K  /var/crash
mypool/var/log        178K  93.2G   178K  /var/log
mypool/var/mail       144K  93.2G   144K  /var/mail
mypool/var/tmp        152K  93.2G   152K  /var/tmp
```

In modern versions of ZFS, `zfs destroy` is asynchronous, and the free space might take minutes to appear in the pool. Use `zpool get freeing poolname` to see the `freeing` property, that shows which datasets are having their blocks freed in the background. If there are child datasets, like snapshots or other datasets, destroying the parent is impossible. To destroy a dataset and its children, use `-r` to recursively destroy the dataset and its children. Use `-n -v` to list datasets and snapshots destroyed by this operation, without actually destroy anything. Space reclaimed by destroying snapshots is also shown.

23.4.2. Creating and Destroying Volumes

A volume is a special dataset type. Rather than mounting as a file system, expose it as a block device under /dev/zvol/poolname/dataset. This allows using the volume for other file systems, to back the disks of a virtual machine, or to make it available to other network hosts using protocols like iSCSI or HAST.

Format a volume with any file system or without a file system to store raw data. To the user, a volume appears to be a regular disk. Putting ordinary file systems on these *zvols* provides features that ordinary disks or file systems do not have. For example, using the compression property on a 250 MB volume allows creation of a compressed FAT file system.

```
# zfs create -V 250m -o compression=on tank/fat32
# zfs list tank
NAME USED AVAIL REFER MOUNTPOINT
tank 258M  670M   31K /tank
# newfs_msdos -F32 /dev/zvol/tank/fat32
# mount -t msdosfs /dev/zvol/tank/fat32 /mnt
# df -h /mnt | grep fat32
Filesystem           Size Used Avail Capacity Mounted on
/dev/zvol/tank/fat32 249M  24k  249M     0%   /mnt
# mount | grep fat32
/dev/zvol/tank/fat32 on /mnt (msdosfs, local)
```

Destroying a volume is much the same as destroying a regular file system dataset. The operation is nearly instantaneous, but it may take minutes to reclaim the free space in the background.

23.4.3. Renaming a Dataset

To change the name of a dataset, use `zfs rename`. To change the parent of a dataset, use this command as well. Renaming a dataset to have a different parent dataset will change the value of those properties inherited from the parent dataset. Renaming a dataset unmounts then remounts it in the new location (inherited from the new parent dataset). To prevent this behavior, use `-u`.

Rename a dataset and move it to be under a different parent dataset:

```
# zfs list
NAME                       USED  AVAIL  REFER  MOUNTPOINT
mypool                     780M  93.2G   144K  none
mypool/ROOT                777M  93.2G   144K  none
mypool/ROOT/default        777M  93.2G   777M  /
mypool/tmp                 176K  93.2G   176K  /tmp
mypool/usr                 704K  93.2G   144K  /usr
mypool/usr/home            184K  93.2G   184K  /usr/home
mypool/usr/mydataset      87.5K  93.2G  87.5K  /usr/mydataset
mypool/usr/ports           144K  93.2G   144K  /usr/ports
```

```
mypool/usr/src         144K  93.2G   144K  /usr/src
mypool/var            1.21M  93.2G   614K  /var
mypool/var/crash       148K  93.2G   148K  /var/crash
mypool/var/log         178K  93.2G   178K  /var/log
mypool/var/mail        144K  93.2G   144K  /var/mail
mypool/var/tmp         152K  93.2G   152K  /var/tmp
# zfs rename mypool/usr/mydataset mypool/var/newname
# zfs list
NAME                  USED  AVAIL  REFER  MOUNTPOINT
mypool                780M  93.2G   144K  none
mypool/ROOT           777M  93.2G   144K  none
mypool/ROOT/default   777M  93.2G   777M  /
mypool/tmp            176K  93.2G   176K  /tmp
mypool/usr            616K  93.2G   144K  /usr
mypool/usr/home       184K  93.2G   184K  /usr/home
mypool/usr/ports      144K  93.2G   144K  /usr/ports
mypool/usr/src        144K  93.2G   144K  /usr/src
mypool/var           1.29M  93.2G   614K  /var
mypool/var/crash      148K  93.2G   148K  /var/crash
mypool/var/log        178K  93.2G   178K  /var/log
mypool/var/mail       144K  93.2G   144K  /var/mail
mypool/var/newname   87.5K  93.2G  87.5K  /var/newname
mypool/var/tmp        152K  93.2G   152K  /var/tmp
```

Renaming snapshots uses the same command. Due to the nature of snapshots, rename cannot change their parent dataset. To rename a recursive snapshot, specify `-r`; this will also rename all snapshots with the same name in child datasets.

```
# zfs list -t snapshot
NAME                                USED  AVAIL  REFER  MOUNTPOINT
mypool/var/newname@first_snapshot      0      -  87.5K  -
# zfs rename mypool/var/newname@first_snapshot new_snapshot_name
# zfs list -t snapshot
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool/var/newname@new_snapshot_name      0      -  87.5K  -
```

23.4.4. Setting Dataset Properties

Each ZFS dataset has properties that control its behavior. Most properties are automatically inherited from the parent dataset, but can be overridden locally. Set a property on a dataset with `zfs set property=value dataset`. Most properties have a limited set of valid values, `zfs get` will display each possible property and valid values. Using `zfs inherit` reverts most properties to their inherited values. User-defined properties are also possible. They become part of the dataset configuration and provide further information about the dataset or its contents. To distinguish these custom properties from the ones supplied as part of ZFS, use a colon (`:`) to create a custom namespace for the property.

```
# zfs set custom:costcenter=1234 tank
# zfs get custom:costcenter tank
NAME PROPERTY           VALUE  SOURCE
tank custom:costcenter  1234   local
```

To remove a custom property, use `zfs inherit` with `-r`. If the custom property is not defined in any of the parent datasets, this option removes it (but the pool's history still records the change).

```
# zfs inherit -r custom:costcenter tank
# zfs get custom:costcenter tank
NAME    PROPERTY           VALUE              SOURCE
tank    custom:costcenter  -                  -
# zfs get all tank | grep custom:costcenter
#
```

Getting and Setting Share Properties

Two commonly used and useful dataset properties are the NFS and SMB share options. Setting these defines if and how ZFS shares datasets on the network. At present, FreeBSD supports setting NFS sharing alone. To get the current status of a share, enter:

```
# zfs get sharenfs mypool/usr/home
NAME             PROPERTY  VALUE    SOURCE
mypool/usr/home  sharenfs  on       local
# zfs get sharesmb mypool/usr/home
NAME             PROPERTY  VALUE    SOURCE
mypool/usr/home  sharesmb  off      local
```

To enable sharing of a dataset, enter:

```
#  zfs set sharenfs=on mypool/usr/home
```

Set other options for sharing datasets through NFS, such as `-alldirs`, `-maproot` and `-network`. To set options on a dataset shared through NFS, enter:

```
#  zfs set sharenfs="-alldirs,-maproot=root,-network=192.168.1.0/24" mypool/usr/home
```

23.4.5. Managing Snapshots

Snapshots are one of the most powerful features of ZFS. A snapshot provides a read-only, point-in-time copy of the dataset. With Copy-On-Write (COW), ZFS creates snapshots fast by preserving older versions of the data on disk. If no snapshots exist, ZFS reclaims space for future use when data is rewritten or deleted. Snapshots preserve disk space by recording just the differences between the current dataset and a previous version. Allowing snapshots on whole datasets, not on individual files or directories. A snapshot from a dataset duplicates everything contained in it. This includes the file system properties, files, directories, permissions, and so on. Snapshots use no extra space when first created, but consume space as the blocks they reference change. Recursive snapshots taken with `-r` create snapshots with the same name on the dataset and its children, providing a consistent moment-in-time snapshot of the file systems. This can be important when an application has files on related datasets or that depend upon each other. Without snapshots, a backup would have copies of the files from different points in time.

Snapshots in ZFS provide a variety of features that even other file systems with snapshot functionality lack. A typical example of snapshot use is as a quick way of backing up the current state of the file system when performing a risky action like a software installation or a system upgrade. If the action fails, rolling back to the snapshot returns the system to the same state when creating the snapshot. If the upgrade was successful, delete the snapshot to free up space. Without snapshots, a failed upgrade often requires restoring backups, which is tedious, time consuming, and may require downtime during which the system is unusable. Rolling back to snapshots is fast, even while the system is running in normal operation, with little or no downtime. The time savings are enormous with multi-terabyte storage systems considering the time required to copy the data from backup. Snapshots are not a replacement for a complete backup of a pool, but offer a quick and easy way to store a dataset copy at a specific time.

Creating Snapshots

To create snapshots, use `zfs snapshot dataset@snapshotname`. Adding `-r` creates a snapshot recursively, with the same name on all child datasets.

Create a recursive snapshot of the entire pool:

```
# zfs list -t all
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool                                 780M  93.2G   144K  none
mypool/ROOT                            777M  93.2G   144K  none
mypool/ROOT/default                    777M  93.2G   777M  /
mypool/tmp                             176K  93.2G   176K  /tmp
mypool/usr                             616K  93.2G   144K  /usr
mypool/usr/home                        184K  93.2G   184K  /usr/home
mypool/usr/ports                       144K  93.2G   144K  /usr/ports
mypool/usr/src                         144K  93.2G   144K  /usr/src
mypool/var                            1.29M  93.2G   616K  /var
mypool/var/crash                       148K  93.2G   148K  /var/crash
mypool/var/log                         178K  93.2G   178K  /var/log
mypool/var/mail                        144K  93.2G   144K  /var/mail
mypool/var/newname                    87.5K  93.2G  87.5K  /var/newname
mypool/var/newname@new_snapshot_name      0      -  87.5K  -
mypool/var/tmp                         152K  93.2G   152K  /var/tmp
# zfs snapshot -r mypool@my_recursive_snapshot
# zfs list -t snapshot
NAME                                        USED  AVAIL  REFER  MOUNTPOINT
mypool@my_recursive_snapshot                   0      -   144K  -
mypool/ROOT@my_recursive_snapshot              0      -   144K  -
mypool/ROOT/default@my_recursive_snapshot      0      -   777M  -
mypool/tmp@my_recursive_snapshot               0      -   176K  -
mypool/usr@my_recursive_snapshot               0      -   144K  -
```

```
mypool/usr/home@my_recursive_snapshot         0      -   184K  -
mypool/usr/ports@my_recursive_snapshot        0      -   144K  -
mypool/usr/src@my_recursive_snapshot          0      -   144K  -
mypool/var@my_recursive_snapshot              0      -   616K  -
mypool/var/crash@my_recursive_snapshot        0      -   148K  -
mypool/var/log@my_recursive_snapshot          0      -   178K  -
mypool/var/mail@my_recursive_snapshot         0      -   144K  -
mypool/var/newname@new_snapshot_name          0      -  87.5K  -
mypool/var/newname@my_recursive_snapshot      0      -  87.5K  -
mypool/var/tmp@my_recursive_snapshot          0      -   152K  -
```

Snapshots are not shown by a normal `zfs list` operation. To list snapshots, append `-t snapshot` to `zfs list`. `-t all` displays both file systems and snapshots.

Snapshots are not mounted directly, showing no path in the `MOUNTPOINT` column. ZFS does not mention available disk space in the `AVAIL` column, as snapshots are read-only after their creation. Compare the snapshot to the original dataset:

```
# zfs list -rt all mypool/usr/home
NAME                                    USED  AVAIL  REFER  MOUNTPOINT
mypool/usr/home                         184K  93.2G   184K  /usr/home
mypool/usr/home@my_recursive_snapshot      0      -   184K  -
```

Displaying both the dataset and the snapshot together reveals how snapshots work in COW fashion. They save the changes (*delta*) made and not the complete file system contents all over again. This means that snapshots take little space when making changes. Observe space usage even more by copying a file to the dataset, then creating a second snapshot:

```
# cp /etc/passwd /var/tmp
# zfs snapshot mypool/var/tmp@after_cp
# zfs list -rt all mypool/var/tmp
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool/var/tmp                         206K  93.2G   118K  /var/tmp
mypool/var/tmp@my_recursive_snapshot    88K      -   152K  -
mypool/var/tmp@after_cp                   0      -   118K  -
```

The second snapshot contains the changes to the dataset after the copy operation. This yields enormous space savings. Notice that the size of the snapshot `mypool/var/tmp@my_recursive_snapshot` also changed in the `USED` column to show the changes between itself and the snapshot taken afterwards.

Comparing Snapshots

ZFS provides a built-in command to compare the differences in content between two snapshots. This is helpful with a lot of snapshots taken over time when the user wants to see how the file system has changed over time. For example, `zfs diff` lets a user find the latest snapshot that still contains a file deleted by accident. Doing this for the two snapshots created in the previous section yields this output:

```
# zfs list -rt all mypool/var/tmp
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool/var/tmp                         206K  93.2G   118K  /var/tmp
mypool/var/tmp@my_recursive_snapshot    88K      -   152K  -
mypool/var/tmp@after_cp                   0      -   118K  -
# zfs diff mypool/var/tmp@my_recursive_snapshot
M       /var/tmp/
+       /var/tmp/passwd
```

The command lists the changes between the specified snapshot (in this case `mypool/var/tmp@my_recursive_snapshot`) and the live file system. The first column shows the change type:

+	Adding the path or file.
-	Deleting the path or file.
M	Modifying the path or file.
R	Renaming the path or file.

Comparing the output with the table, it becomes clear that ZFS added passwd after creating the snapshot `mypool/var/tmp@my_recursive_snapshot`. This also resulted in a modification to the parent directory mounted at `/var/tmp`.

Comparing two snapshots is helpful when using the ZFS replication feature to transfer a dataset to a different host for backup purposes.

Compare two snapshots by providing the full dataset name and snapshot name of both datasets:

```
# cp /var/tmp/passwd /var/tmp/passwd.copy
# zfs snapshot mypool/var/tmp@diff_snapshot
# zfs diff mypool/var/tmp@my_recursive_snapshot mypool/var/tmp@diff_snapshot
M       /var/tmp/
+       /var/tmp/passwd
+       /var/tmp/passwd.copy
# zfs diff mypool/var/tmp@my_recursive_snapshot mypool/var/tmp@after_cp
M       /var/tmp/
+       /var/tmp/passwd
```

A backup administrator can compare two snapshots received from the sending host and determine the actual changes in the dataset. See the Replication section for more information.

Snapshot Rollback

When at least one snapshot is available, roll back to it at any time. Most often this is the case when the current state of the dataset is no longer valid or an older version is preferred. Scenarios such as local development tests gone wrong, botched system updates hampering the system functionality, or the need to restore deleted files or directories are all too common occurrences. To roll back a snapshot, use `zfs rollback snapshotname`. If a lot of changes are present, the operation will take a long time. During that time, the dataset always remains in a consistent state, much like a database that conforms to ACID principles is performing a rollback. This is happening while the dataset is live and accessible without requiring a downtime. Once the snapshot rolled back, the dataset has the same state as it had when the snapshot was originally taken. Rolling back to a snapshot discards all other data in that dataset not part of the snapshot. Taking a snapshot of the current state of the dataset before rolling back to a previous one is a good idea when requiring some data later. This way, the user can roll back and forth between snapshots without losing data that is still valuable.

In the first example, roll back a snapshot because a careless `rm` operation removed more data than intended.

```
# zfs list -rt all mypool/var/tmp
NAME                                    USED  AVAIL  REFER  MOUNTPOINT
mypool/var/tmp                          262K  93.2G   120K  /var/tmp
mypool/var/tmp@my_recursive_snapshot     88K      -   152K  -
mypool/var/tmp@after_cp                53.5K      -   118K  -
mypool/var/tmp@diff_snapshot               0      -   120K  -
# ls /var/tmp
passwd          passwd.copy     vi.recover
# rm /var/tmp/passwd*
# ls /var/tmp
vi.recover
```

At this point, the user notices the removal of extra files and wants them back. ZFS provides an easy way to get them back using rollbacks, when performing snapshots of important data on a regular basis. To get the files back and start over from the last snapshot, issue the command:

```
# zfs rollback mypool/var/tmp@diff_snapshot
# ls /var/tmp
passwd          passwd.copy     vi.recover
```

The rollback operation restored the dataset to the state of the last snapshot. Rolling back to a snapshot taken much earlier with other snapshots taken afterwards is also possible. When trying to do this, ZFS will issue this warning:

```
# zfs list -rt snapshot mypool/var/tmp
AME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool/var/tmp@my_recursive_snapshot    88K      -   152K  -
mypool/var/tmp@after_cp              53.5K      -   118K  -
mypool/var/tmp@diff_snapshot             0      -   120K  -
# zfs rollback mypool/var/tmp@my_recursive_snapshot
cannot rollback to 'mypool/var/tmp@my_recursive_snapshot': more recent snapshots exist
use '-r' to force deletion of the following snapshots:
mypool/var/tmp@after_cp
```

```
mypool/var/tmp@diff_snapshot
```

This warning means that snapshots exist between the current state of the dataset and the snapshot to which the user wants to roll back. To complete the rollback delete these snapshots. ZFS cannot track all the changes between different states of the dataset, because snapshots are read-only. ZFS will not delete the affected snapshots unless the user specifies `-r` to confirm that this is the desired action. If that is the intention, and understanding the consequences of losing all intermediate snapshots, issue the command:

```
# zfs rollback -r mypool/var/tmp@my_recursive_snapshot
# zfs list -rt snapshot mypool/var/tmp
NAME                                   USED  AVAIL  REFER  MOUNTPOINT
mypool/var/tmp@my_recursive_snapshot     8K      -   152K  -
# ls /var/tmp
vi.recover
```

The output from `zfs list -t snapshot` confirms the removal of the intermediate snapshots as a result of `zfs rollback -r`.

Restoring Individual Files from Snapshots

Snapshots live in a hidden directory under the parent dataset: .zfs/snapshots/snapshotname. By default, these directories will not show even when executing a standard `ls -a` . Although the directory doesn't show, access it like any normal directory. The property named `snapdir` controls whether these hidden directories show up in a directory listing. Setting the property to `visible` allows them to appear in the output of `ls` and other commands that deal with directory contents.

```
# zfs get snapdir mypool/var/tmp
NAME            PROPERTY  VALUE    SOURCE
mypool/var/tmp  snapdir   hidden   default
# ls -a /var/tmp
.               ..              passwd          vi.recover
# zfs set snapdir=visible mypool/var/tmp
# ls -a /var/tmp
.               ..              .zfs            passwd          vi.recover
```

Restore individual files to a previous state by copying them from the snapshot back to the parent dataset. The directory structure below .zfs/snapshot has a directory named like the snapshots taken earlier to make it easier to identify them. The next example shows how to restore a file from the hidden .zfs directory by copying it from the snapshot containing the latest version of the file:

```
# rm /var/tmp/passwd
# ls -a /var/tmp
.               ..              .zfs            vi.recover
# ls /var/tmp/.zfs/snapshot
after_cp                my_recursive_snapshot
# ls /var/tmp/.zfs/snapshot/after_cp
```

```
passwd          vi.recover
# cp /var/tmp/.zfs/snapshot/after_cp/passwd /var/tmp
```

Even if the `snapdir` property is set to hidden, running `ls .zfs/snapshot` will still list the contents of that directory. The administrator decides whether to display these directories. This is a per-dataset setting. Copying files or directories from this hidden .zfs/snapshot is simple enough. Trying it the other way around results in this error:

```
# cp /etc/rc.conf /var/tmp/.zfs/snapshot/after_cp/
cp: /var/tmp/.zfs/snapshot/after_cp/rc.conf: Read-only file system
```

The error reminds the user that snapshots are read-only and cannot change after creation. Copying files into and removing them from snapshot directories are both disallowed because that would change the state of the dataset they represent.

Snapshots consume space based on how much the parent file system has changed since the time of the snapshot. The `written` property of a snapshot tracks the space the snapshot uses.

To destroy snapshots and reclaim the space, use `zfs destroy dataset@snapshot`. Adding `-r` recursively removes all snapshots with the same name under the parent dataset. Adding `-n -v` to the command displays a list of the snapshots to be deleted and an estimate of the space it would reclaim without performing the actual destroy operation.

23.4.6. Managing Clones

A clone is a copy of a snapshot treated more like a regular dataset. Unlike a snapshot, a clone is writeable and mountable, and has its own properties. After creating a clone using `zfs clone`, destroying the originating snapshot is impossible. To reverse the child/parent relationship between the clone and the snapshot use `zfs promote`. Promoting a clone makes the snapshot become a child of the clone, rather than of the original parent dataset. This will change how ZFS accounts for the space, but not actually change the amount of space consumed. Mounting the clone anywhere within the ZFS file system hierarchy is possible, not only below the original location of the snapshot.

To show the clone feature use this example dataset:

```
# zfs list -rt all camino/home/joe
NAME                    USED  AVAIL  REFER  MOUNTPOINT
camino/home/joe         108K   1.3G    87K  /usr/home/joe
camino/home/joe@plans    21K      -  85.5K  -
camino/home/joe@backup    0K      -    87K  -
```

A typical use for clones is to experiment with a specific dataset while keeping the snapshot around to fall back to in case something goes wrong. Since snapshots cannot change, create a read/write clone of a snapshot. After achieving the desired result in the clone, promote the clone to a dataset and remove the old file system. Removing the parent dataset is not strictly necessary, as the clone and dataset can coexist without problems.

```
# zfs clone camino/home/joe@backup camino/home/joenew
# ls /usr/home/joe*
/usr/home/joe:
backup.txz     plans.txt

/usr/home/joenew:
backup.txz     plans.txt
# df -h /usr/home
Filesystem          Size    Used   Avail Capacity  Mounted on
usr/home/joe        1.3G     31k    1.3G     0%    /usr/home/joe
usr/home/joenew     1.3G     31k    1.3G     0%    /usr/home/joenew
```

Creating a clone makes it an exact copy of the state the dataset was in when taking the snapshot. Changing the clone independently from its originating dataset is possible now. The connection between the two is the snapshot. ZFS records this connection in the property `origin`. Promoting the clone with `zfs promote` makes the clone an independent dataset. This removes the value of the `origin` property and disconnects the newly independent dataset from the snapshot. This example shows it:

```
# zfs get origin camino/home/joenew
NAME                  PROPERTY  VALUE                     SOURCE
camino/home/joenew    origin    camino/home/joe@backup    -
# zfs promote camino/home/joenew
# zfs get origin camino/home/joenew
NAME                  PROPERTY  VALUE   SOURCE
camino/home/joenew    origin    -       -
```

After making some changes like copying loader.conf to the promoted clone, for example, the old directory becomes obsolete in this case. Instead, the promoted clone can replace it. To do this, `zfs destroy` the old dataset first and then `zfs rename` the clone to the old dataset name (or to an entirely different name).

```
# cp /boot/defaults/loader.conf /usr/home/joenew
# zfs destroy -f camino/home/joe
# zfs rename camino/home/joenew camino/home/joe
# ls /usr/home/joe
backup.txz     loader.conf     plans.txt
# df -h /usr/home
Filesystem          Size    Used   Avail Capacity  Mounted on
usr/home/joe        1.3G    128k    1.3G     0%    /usr/home/joe
```

The cloned snapshot is now an ordinary dataset. It contains all the data from the original snapshot plus the files added to it like loader.conf. Clones provide useful features to ZFS users in different scenarios. For example, provide jails as snapshots containing different sets of installed applications. Users can clone these snapshots and add their own applications as they see fit. Once satisfied with the changes, promote the clones to full datasets and provide them to end users to work with like they would with a real dataset. This saves time and administrative overhead when providing these jails.

23.4.7. Replication

Keeping data on a single pool in one location exposes it to risks like theft and natural or human disasters. Making regular backups of the entire pool is vital. ZFS provides a built-in serialization feature that can send a stream representation of the data to standard output. Using this feature, storing this data on another pool connected to the local system is possible, as is sending it over a network to another system. Snapshots are the basis for this replication (see the section on ZFS snapshots). The commands used for replicating data are `zfs send` and `zfs receive`.

These examples show ZFS replication with these two pools:

```
# zpool list
NAME    SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG   CAP  DEDUP  HEALTH  ALTROOT
backup  960M    77K   896M         -         -     0%    0%  1.00x  ONLINE  -
mypool  984M  43.7M   940M         -         -     0%    4%  1.00x  ONLINE  -
```

The pool named *mypool* is the primary pool where writing and reading data happens on a regular basis. Using a second standby pool *backup* in case the primary pool becomes unavailable. Note that this fail-over is not done automatically by ZFS, but must be manually done by a system administrator when needed. Use a snapshot to provide a consistent file system version to replicate. After creating a snapshot of *mypool*, copy it to the *backup* pool by replicating snapshots. This does not include changes made since the most recent snapshot.

```
# zfs snapshot mypool@backup1
# zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
mypool@backup1             0      -  43.6M  -
```

Now that a snapshot exists, use `zfs send` to create a stream representing the contents of the snapshot. Store this stream as a file or receive it on another pool. Write the stream to standard output, but redirect to a file or pipe or an error appears:

```
# zfs send mypool@backup1
Error: Stream can not be written to a terminal.
You must redirect standard output.
```

To back up a dataset with `zfs send`, redirect to a file located on the mounted backup pool. Ensure that the pool has enough free space to accommodate the size of the sent snapshot, which means the data contained in the snapshot, not the changes from the previous snapshot.

```
# zfs send mypool@backup1 > /backup/backup1
# zpool list
NAME    SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
```

```
backup  960M  63.7M   896M         -         -     0%     6%  1.00x  ONLINE  -
mypool  984M  43.7M   940M         -         -     0%     4%  1.00x  ONLINE  -
```

The `zfs send` transferred all the data in the snapshot called *backup1* to the pool named *backup*. To create and send these snapshots automatically, use a cron(8) job.

Instead of storing the backups as archive files, ZFS can receive them as a live file system, allowing direct access to the backed up data. To get to the actual data contained in those streams, use `zfs receive` to transform the streams back into files and directories. The example below combines `zfs send` and `zfs receive` using a pipe to copy the data from one pool to another. Use the data directly on the receiving pool after the transfer is complete. It is only possible to replicate a dataset to an empty dataset.

```
# zfs snapshot mypool@replica1
# zfs send -v mypool@replica1 | zfs receive backup/mypool
send from @ to mypool@replica1 estimated size is 50.1M
total estimated size is 50.1M
TIME        SENT    SNAPSHOT

# zpool list
NAME    SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG    CAP  DEDUP  HEALTH  ALTROOT
backup  960M  63.7M   896M         -         -     0%     6%  1.00x  ONLINE  -
mypool  984M  43.7M   940M         -         -     0%     4%  1.00x  ONLINE  -
```

Incremental Backups

`zfs send` can also determine the difference between two snapshots and send individual differences between the two. This saves disk space and transfer time. For example:

```
# zfs snapshot mypool@replica2
# zfs list -t snapshot
NAME                    USED  AVAIL  REFER  MOUNTPOINT
mypool@replica1         5.72M      -  43.6M  -
mypool@replica2             0      -  44.1M  -
# zpool list
NAME    SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG   CAP  DEDUP  HEALTH  ALTROOT
backup  960M  61.7M   898M         -         -     0%    6%  1.00x  ONLINE  -
mypool  960M  50.2M   910M         -         -     0%    5%  1.00x  ONLINE  -
```

Create a second snapshot called *replica2*. This second snapshot contains changes made to the file system between now and the previous snapshot, *replica1*. Using `zfs send -i` and indicating the pair of snapshots generates an incremental replica stream containing the changed data. This succeeds if the initial snapshot already exists on the receiving side.

```
# zfs send -v -i mypool@replica1 mypool@replica2 | zfs receive /backup/mypool
send from @replica1 to mypool@replica2 estimated size is 5.02M
total estimated size is 5.02M
```

```
TIME        SENT   SNAPSHOT

# zpool list
NAME    SIZE  ALLOC   FREE   CKPOINT  EXPANDSZ   FRAG  CAP  DEDUP  HEALTH  ALTROOT
backup  960M  80.8M   879M         -         -     0%   8%  1.00x  ONLINE  -
mypool  960M  50.2M   910M         -         -     0%   5%  1.00x  ONLINE  -

# zfs list
NAME                         USED  AVAIL  REFER  MOUNTPOINT
backup                      55.4M   240G   152K  /backup
backup/mypool               55.3M   240G  55.2M  /backup/mypool
mypool                      55.6M  11.6G  55.0M  /mypool

# zfs list -t snapshot
NAME                                          USED  AVAIL  REFER  MOUNTPOINT
backup/mypool@replica1                        104K      -  50.2M  -
backup/mypool@replica2                           0      -  55.2M  -
mypool@replica1                              29.9K      -  50.0M  -
mypool@replica2                                  0      -  55.0M  -
```

The incremental stream replicated the changed data rather than the entirety of *replica1*. Sending the differences alone took much less time to transfer and saved disk space by not copying the whole pool each time. This is useful when replicating over a slow network or one charging per transferred byte.

A new file system, *backup/mypool*, is available with the files and data from the pool *mypool*. Specifying `-p` copies the dataset properties including compression settings, quotas, and mount points. Specifying `-R` copies all child datasets of the dataset along with their properties. Automate sending and receiving to create regular backups on the second pool.

Sending Encrypted Backups over SSH

Sending streams over the network is a good way to keep a remote backup, but it does come with a drawback. Data sent over the network link is not encrypted, allowing anyone to intercept and transform the streams back into data without the knowledge of the sending user. This is undesirable when sending the streams over the internet to a remote host. Use SSH to securely encrypt data sent over a network connection. Since ZFS requires redirecting the stream from standard output, piping it through SSH is easy. To keep the contents of the file system encrypted in transit and on the remote system, consider using PEFS.

Change some settings and take security precautions first. This describes the necessary steps required for the `zfs send` operation; for more information on SSH, see OpenSSH.

Change the configuration as follows:

- Passwordless SSH access between sending and receiving host using SSH keys
- ZFS requires the privileges of the `root` user to send and receive streams. This requires logging in to the receiving system as `root`.
- Security reasons prevent `root` from logging in by default.

- Use the ZFS Delegation system to allow a non-`root` user on each system to perform the respective send and receive operations. On the sending system:

```
# zfs allow -u someuser send,snapshot mypool
```

- To mount the pool, the unprivileged user must own the directory, and regular users need permission to mount file systems.

On the receiving system:

```
# sysctl vfs.usermount=1
vfs.usermount: 0 -> 1
# echo vfs.usermount=1 >> /etc/sysctl.conf
# zfs create recvpool/backup
# zfs allow -u someuser create,mount,receive recvpool/backup
# chown someuser /recvpool/backup
```

The unprivileged user can receive and mount datasets now, and replicates the *home* dataset to the remote system:

```
% zfs snapshot -r mypool/home@monday
% zfs send -R mypool/home@monday | ssh someuser@backuphost zfs recv -dvu
recvpool/backup
```

Create a recursive snapshot called *monday* of the file system dataset *home* on the pool *mypool*. Then `zfs send -R` includes the dataset, all child datasets, snapshots, clones, and settings in the stream. Pipe the output through SSH to the waiting `zfs receive` on the remote host *backuphost*. Using an IP address or fully qualified domain name is good practice. The receiving machine writes the data to the *backup* dataset on the *recvpool* pool. Adding `-d` to `zfs recv` overwrites the name of the pool on the receiving side with the name of the snapshot. `-u` causes the file systems to not mount on the receiving side. Using `-v` shows more details about the transfer, including the elapsed time and the amount of data transferred.

23.4.8. Dataset, User, and Group Quotas

Use Dataset quotas to restrict the amount of space consumed by a particular dataset. Reference Quotas work in much the same way, but count the space used by the dataset itself, excluding snapshots and child datasets. Similarly, use user and group quotas to prevent users or groups from using up all the space in the pool or dataset.

The following examples assume that the users already exist in the system. Before adding a user to the system, make sure to create their home dataset first and set the `mountpoint` to `/home/bob`. Then, create the user and make the home directory point to the dataset's `mountpoint` location. This will properly set owner and group permissions without shadowing any pre-existing home directory paths that might exist.

To enforce a dataset quota of 10 GB for storage/home/bob:

```
# zfs set quota=10G storage/home/bob
```

To enforce a reference quota of 10 GB for storage/home/bob:

```
# zfs set refquota=10G storage/home/bob
```

To remove a quota of 10 GB for storage/home/bob:

```
# zfs set quota=none storage/home/bob
```

The general format is `userquota@user=size`, and the user's name must be in one of these formats:

- POSIX compatible name such as *joe*.
- POSIX numeric ID such as *789*.
- SID name such as *joe.bloggs@example.com*.
- SID numeric ID such as *S-1-123-456-789*.

For example, to enforce a user quota of 50 GB for the user named *joe*:

```
# zfs set userquota@joe=50G
```

To remove any quota:

```
# zfs set userquota@joe=none
```

User quota properties are not displayed by `zfs get all`. Non-`root` users can't see other's quotas unless granted the `userquota` privilege. Users with this privilege are able to view and set everyone's quota.

The general format for setting a group quota is: `groupquota@group=size`.

To set the quota for the group *firstgroup* to 50 GB, use:

```
# zfs set groupquota@firstgroup=50G
```

To remove the quota for the group *firstgroup*, or to make sure that one is not set, instead use:

```
# zfs set groupquota@firstgroup=none
```

As with the user quota property, non-`root` users can see the quotas associated with the groups to which they belong. A user with the `groupquota` privilege or `root` can view and set all quotas for all groups.

To display the amount of space used by each user on a file system or snapshot along with any quotas, use `zfs userspace`. For group information, use `zfs groupspace`. For more information about supported options or how to display specific options alone, refer to zfs(1).

Privileged users and `root` can list the quota for storage/home/bob using:

```
# zfs get quota storage/home/bob
```

23.4.9. Reservations

Reservations guarantee an always-available amount of space on a dataset. The reserved space will not be available to any other dataset. This useful feature ensures that free space is available for an important dataset or log files.

The general format of the `reservation` property is `reservation=size`, so to set a reservation of 10 GB on storage/home/bob, use:

```
# zfs set reservation=10G storage/home/bob
```

To clear any reservation:

```
# zfs set reservation=none storage/home/bob
```

The same principle applies to the `refreservation` property for setting a Reference Reservation, with the general format `refreservation=size`.

This command shows any reservations or refreservations that exist on storage/home/bob:

```
# zfs get reservation storage/home/bob
# zfs get refreservation storage/home/bob
```

23.4.10. Compression

ZFS provides transparent compression. Compressing data written at the block level saves space and also increases disk throughput. If data compresses by 25% the compressed data writes to the disk at the same rate as the uncompressed version, resulting in an effective write speed of 125%. Compression can also be a great alternative to Deduplication because it does not require extra memory.

ZFS offers different compression algorithms, each with different trade-offs. The introduction of LZ4 compression in ZFS v5000 enables compressing the entire pool without the large performance trade-off of other algorithms. The biggest advantage to LZ4 is the *early abort* feature. If LZ4 does not achieve at least 12.5% compression in the header part of the data, ZFS writes the block uncompressed to avoid wasting CPU cycles trying to compress data that is either already compressed or uncompressible. For details about the different compression algorithms available in ZFS, see the Compression entry in the terminology section.

The administrator can see the effectiveness of compression using dataset properties.

```
# zfs get used,compressratio,compression,logicalused mypool/compressed_dataset
NAME        PROPERTY          VALUE     SOURCE
mypool/compressed_dataset  used              449G      -
mypool/compressed_dataset  compressratio     1.11x     -
mypool/compressed_dataset  compression       lz4       local
mypool/compressed_dataset  logicalused       496G      -
```

The dataset is using 449 GB of space (the used property). Without compression, it would have taken 496 GB of space (the `logicalused` property). This results in a 1.11:1 compression ratio.

Compression can have an unexpected side effect when combined with User Quotas. User quotas restrict how much actual space a user consumes on a dataset *after compression*. If a user has a quota of 10 GB, and writes 10 GB of compressible data, they will still be able to store more data. If they later update a file, say a database, with more or less compressible data, the amount of space available to them will change. This can result in the odd situation where a user did not increase the actual amount of data (the `logicalused` property), but the change in compression caused them to reach their quota limit.

Compression can have a similar unexpected interaction with backups. Quotas are often used to limit data storage to ensure there is enough backup space available. Since quotas do not consider compression ZFS may write more data than would fit with uncompressed backups.

23.4.11. Zstandard Compression

OpenZFS 2.0 added a new compression algorithm. Zstandard (Zstd) offers higher compression ratios than the default LZ4 while offering much greater speeds than the alternative, gzip. OpenZFS 2.0 is available starting with FreeBSD 12.1-RELEASE via sysutils/openzfs and has been the default since FreeBSD 13.0-RELEASE.

Zstd provides a large selection of compression levels, providing fine-grained control over performance versus compression ratio. One of the main advantages of Zstd is that the decompression speed is independent of the compression level. For data written once but read often, Zstd allows the use of the highest compression levels without a read performance penalty.

Even with frequent data updates, enabling compression often provides higher performance. One of the biggest advantages comes from the compressed ARC feature. ZFS's Adaptive Replacement Cache (ARC) caches the compressed version of the data in RAM, decompressing it each time. This allows the same amount of RAM to store more data and metadata, increasing the cache hit ratio.

ZFS offers 19 levels of Zstd compression, each offering incrementally more space savings in exchange for slower compression. The default level is `zstd-3` and offers greater compression than

LZ4 without being much slower. Levels above 10 require large amounts of memory to compress each block and systems with less than 16 GB of RAM should not use them. ZFS uses a selection of the Zstd_fast_ levels also, which get correspondingly faster but supports lower compression ratios. ZFS supports `zstd-fast-1` through `zstd-fast-10`, `zstd-fast-20` through `zstd-fast-100` in increments of 10, and `zstd-fast-500` and `zstd-fast-1000` which provide minimal compression, but offer high performance.

If ZFS is not able to get the required memory to compress a block with Zstd, it will fall back to storing the block uncompressed. This is unlikely to happen except at the highest levels of Zstd on memory constrained systems. ZFS counts how often this has occurred since loading the ZFS module with `kstat.zfs.misc.zstd.compress_alloc_fail`.

23.4.12. Deduplication

When enabled, deduplication uses the checksum of each block to detect duplicate blocks. When a new block is a duplicate of an existing block, ZFS writes a new reference to the existing data instead of the whole duplicate block. Tremendous space savings are possible if the data contains a lot of duplicated files or repeated information. Warning: deduplication requires a large amount of memory, and enabling compression instead provides most of the space savings without the extra cost.

To activate deduplication, set the `dedup` property on the target pool:

```
# zfs set dedup=on pool
```

Deduplicating only affects new data written to the pool. Merely activating this option will not deduplicate data already written to the pool. A pool with a freshly activated deduplication property will look like this example:

```
# zpool list
NAME  SIZE ALLOC  FREE   CKPOINT  EXPANDSZ   FRAG   CAP   DEDUP   HEALTH   ALTROOT
pool 2.84G 2.19M 2.83G         -         -     0%    0%   1.00x   ONLINE   -
```

The `DEDUP` column shows the actual rate of deduplication for the pool. A value of `1.00x` shows that data has not deduplicated yet. The next example copies some system binaries three times into different directories on the deduplicated pool created above.

```
# for d in dir1 dir2 dir3; do
> mkdir $d && cp -R /usr/bin $d &
> done
```

To observe deduplicating of redundant data, use:

```
# zpool list
NAME SIZE  ALLOC  FREE   CKPOINT  EXPANDSZ   FRAG  CAP   DEDUP   HEALTH   ALTROOT
```

```
pool 2.84G 20.9M 2.82G         -         -     0%    0%    3.00x    ONLINE    -
```

The `DEDUP` column shows a factor of `3.00x`. Detecting and deduplicating copies of the data uses a third of the space. The potential for space savings can be enormous, but comes at the cost of having enough memory to keep track of the deduplicated blocks.

Deduplication is not always beneficial when the data in a pool is not redundant. ZFS can show potential space savings by simulating deduplication on an existing pool:

```
# zdb -S pool
Simulated DDT histogram:

bucket              allocated                       referenced

______   ______________________________   ______________________________
refcnt   blocks   LSIZE   PSIZE   DSIZE   blocks   LSIZE   PSIZE   DSIZE
------   ------   -----   -----   -----   ------   -----   -----   -----
     1    2.58M    289G    264G    264G    2.58M    289G    264G    264G
     2     206K   12.6G   10.4G   10.4G     430K   26.4G   21.6G   21.6G
     4    37.6K    692M    276M    276M     170K   3.04G   1.26G   1.26G
     8    2.18K   45.2M   19.4M   19.4M    20.0K    425M    176M    176M
    16      174   2.83M   1.20M   1.20M    3.33K   48.4M   20.4M   20.4M
    32       40   2.17M    222K    222K    1.70K   97.2M   9.91M   9.91M
    64        9     56K   10.5K   10.5K      865   4.96M    948K    948K
   128        2   9.50K      2K      2K      419   2.11M    438K    438K
   256        5   61.5K     12K     12K    1.90K   23.0M   4.47M   4.47M
    1K        2      1K      1K      1K    2.98K   1.49M   1.49M   1.49M
 Total    2.82M    303G    275G    275G    3.20M    319G    287G    287G

dedup = 1.05, compress = 1.11, copies = 1.00, dedup * compress / copies = 1.16
```

After `zdb -S` finishes analyzing the pool, it shows the space reduction ratio that activating deduplication would achieve. In this case, `1.16` is a poor space saving ratio mainly provided by compression. Activating deduplication on this pool would not save any amount of space, and is not worth the amount of memory required to enable deduplication. Using the formula *ratio = dedup * compress / copies*, system administrators can plan the storage allocation, deciding whether the workload will contain enough duplicate blocks to justify the memory requirements. If the data is reasonably compressible, the space savings may be good. Good practice is to enable compression first as compression also provides greatly increased performance. Enable deduplication in cases where savings are considerable and with enough available memory for the DDT.

23.4.13. ZFS and Jails

Use `zfs jail` and the corresponding `jailed` property to delegate a ZFS dataset to a Jail. `zfs jail jailid` attaches a dataset to the specified jail, and `zfs unjail` detaches it. To control the dataset from within a jail, set the `jailed` property. ZFS forbids mounting a jailed dataset on the host because it may have mount points that would compromise the security of the host.

23.5. Delegated Administration

A comprehensive permission delegation system allows unprivileged users to perform ZFS administration functions. For example, if each user's home directory is a dataset, users need permission to create and destroy snapshots of their home directories. A user performing backups can get permission to use replication features. ZFS allows a usage statistics script to run with access to only the space usage data for all users. Delegating the ability to delegate permissions is also possible. Permission delegation is possible for each subcommand and most properties.

23.5.1. Delegating Dataset Creation

`zfs allow someuser create mydataset` gives the specified user permission to create child datasets under the selected parent dataset. A caveat: creating a new dataset involves mounting it. That requires setting the FreeBSD `vfs.usermount` sysctl(8) to `1` to allow non-root users to mount a file system. Another restriction aimed at preventing abuse: non-`root` users must own the mountpoint where mounting the file system.

23.5.2. Delegating Permission Delegation

`zfs allow someuser allow mydataset` gives the specified user the ability to assign any permission they have on the target dataset, or its children, to other users. If a user has the `snapshot` permission and the `allow` permission, that user can then grant the `snapshot` permission to other users.

23.6. Advanced Topics

23.6.1. Tuning

Adjust tunables to make ZFS perform best for different workloads.

- `vfs.zfs.arc.max` starting with 13.x (`vfs.zfs.arc_max` for 12.x) - Upper size of the ARC. The default is all RAM but 1 GB, or 5/8 of all RAM, whichever is more. Use a lower value if the system runs any other daemons or processes that may require memory. Adjust this value at runtime with sysctl(8) and set it in /boot/loader.conf or /etc/sysctl.conf.
- `vfs.zfs.arc.meta_limit` starting with 13.x (`vfs.zfs.arc_meta_limit` for 12.x) - Limit the amount of the ARC used to store metadata. The default is one fourth of `vfs.zfs.arc.max`. Increasing this value will improve performance if the workload involves operations on a large number of files and directories, or frequent metadata operations, at the cost of less file data fitting in the ARC. Adjust this value at runtime with sysctl(8) in /boot/loader.conf or /etc/sysctl.conf.
- `vfs.zfs.arc.min` starting with 13.x (`vfs.zfs.arc_min` for 12.x) - Lower size of the ARC. The default is one half of `vfs.zfs.arc.meta_limit`. Adjust this value to prevent other applications from pressuring out the entire ARC. Adjust this value at runtime with sysctl(8) and in /boot/loader.conf or /etc/sysctl.conf.
- `vfs.zfs.vdev.cache.size` - A preallocated amount of memory reserved as a cache for each device in the pool. The total amount of memory used will be this value multiplied by the number of devices. Set this value at boot time and in /boot/loader.conf.
- `vfs.zfs.min_auto_ashift` - Lower `ashift` (sector size) used automatically at pool creation time.

The value is a power of two. The default value of `9` represents `2^9 = 512`, a sector size of 512 bytes. To avoid *write amplification* and get the best performance, set this value to the largest sector size used by a device in the pool.

Common drives have 4 KB sectors. Using the default `ashift` of `9` with these drives results in write amplification on these devices. Data contained in a single 4 KB write is instead written in eight 512-byte writes. ZFS tries to read the native sector size from all devices when creating a pool, but drives with 4 KB sectors report that their sectors are 512 bytes for compatibility. Setting `vfs.zfs.min_auto_ashift` to `12` (`2^12 = 4096`) before creating a pool forces ZFS to use 4 KB blocks for best performance on these drives.

Forcing 4 KB blocks is also useful on pools with planned disk upgrades. Future disks use 4 KB sectors, and `ashift` values cannot change after creating a pool.

In some specific cases, the smaller 512-byte block size might be preferable. When used with 512-byte disks for databases or as storage for virtual machines, less data transfers during small random reads. This can provide better performance when using a smaller ZFS record size.

- `vfs.zfs.prefetch.disable` - Disable prefetch. A value of `0` enables and `1` disables it. The default is `0`, unless the system has less than 4 GB of RAM. Prefetch works by reading larger blocks than requested into the ARC in hopes to soon need the data. If the workload has a large number of random reads, disabling prefetch may actually improve performance by reducing unnecessary reads. Adjust this value at any time with sysctl(8).
- `vfs.zfs.vdev.trim_on_init` - Control whether new devices added to the pool have the `TRIM` command run on them. This ensures the best performance and longevity for SSDs, but takes extra time. If the device has already been secure erased, disabling this setting will make the addition of the new device faster. Adjust this value at any time with sysctl(8).
- `vfs.zfs.vdev.max_pending` - Limit the number of pending I/O requests per device. A higher value will keep the device command queue full and may give higher throughput. A lower value will reduce latency. Adjust this value at any time with sysctl(8).
- `vfs.zfs.top_maxinflight` - Upper number of outstanding I/Os per top-level vdev. Limits the depth of the command queue to prevent high latency. The limit is per top-level vdev, meaning the limit applies to each mirror, RAID-Z, or other vdev independently. Adjust this value at any time with sysctl(8).
- `vfs.zfs.l2arc_write_max` - Limit the amount of data written to the L2ARC per second. This tunable extends the longevity of SSDs by limiting the amount of data written to the device. Adjust this value at any time with sysctl(8).
- `vfs.zfs.l2arc_write_boost` - Adds the value of this tunable to `vfs.zfs.l2arc_write_max` and increases the write speed to the SSD until evicting the first block from the L2ARC. This "Turbo Warmup Phase" reduces the performance loss from an empty L2ARC after a reboot. Adjust this value at any time with sysctl(8).
- `vfs.zfs.scrub_delay` - Number of ticks to delay between each I/O during a `scrub`. To ensure that a `scrub` does not interfere with the normal operation of the pool, if any other I/O is happening the `scrub` will delay between each command. This value controls the limit on the total IOPS (I/Os Per Second) generated by the `scrub`. The granularity of the setting is determined by the value of `kern.hz` which defaults to 1000 ticks per second. Changing this setting results in a different

effective IOPS limit. The default value is `4`, resulting in a limit of: 1000 ticks/sec / 4 = 250 IOPS. Using a value of *20* would give a limit of: 1000 ticks/sec / 20 = 50 IOPS. Recent activity on the pool limits the speed of `scrub`, as determined by `vfs.zfs.scan_idle`. Adjust this value at any time with sysctl(8).

- `vfs.zfs.resilver_delay` - Number of milliseconds of delay inserted between each I/O during a resilver. To ensure that a resilver does not interfere with the normal operation of the pool, if any other I/O is happening the resilver will delay between each command. This value controls the limit of total IOPS (I/Os Per Second) generated by the resilver. ZFS determins the granularity of the setting by the value of `kern.hz` which defaults to 1000 ticks per second. Changing this setting results in a different effective IOPS limit. The default value is 2, resulting in a limit of: 1000 ticks/sec / 2 = 500 IOPS. Returning the pool to an Online state may be more important if another device failing could Fault the pool, causing data loss. A value of 0 will give the resilver operation the same priority as other operations, speeding the healing process. Other recent activity on the pool limits the speed of resilver, as determined by `vfs.zfs.scan_idle`. Adjust this value at any time with sysctl(8).
- `vfs.zfs.scan_idle` - Number of milliseconds since the last operation before considering the pool is idle. ZFS disables the rate limiting for `scrub` and resilver when the pool is idle. Adjust this value at any time with sysctl(8).
- `vfs.zfs.txg.timeout` - Upper number of seconds between transaction groups. The current transaction group writes to the pool and a fresh transaction group starts if this amount of time elapsed since the previous transaction group. A transaction group may trigger earlier if writing enough data. The default value is 5 seconds. A larger value may improve read performance by delaying asynchronous writes, but this may cause uneven performance when writing the transaction group. Adjust this value at any time with sysctl(8).

23.6.2. ZFS on i386

Some of the features provided by ZFS are memory intensive, and may require tuning for upper efficiency on systems with limited RAM.

Memory

As a lower value, the total system memory should be at least one gigabyte. The amount of recommended RAM depends upon the size of the pool and which features ZFS uses. A general rule of thumb is 1 GB of RAM for every 1 TB of storage. If using the deduplication feature, a general rule of thumb is 5 GB of RAM per TB of storage to deduplicate. While some users use ZFS with less RAM, systems under heavy load may panic due to memory exhaustion. ZFS may require further tuning for systems with less than the recommended RAM requirements.

Kernel Configuration

Due to the address space limitations of the i386™ platform, ZFS users on the i386™ architecture must add this option to a custom kernel configuration file, rebuild the kernel, and reboot:

```
options        KVA_PAGES=512
```

This expands the kernel address space, allowing the `vm.kvm_size` tunable to push beyond the imposed limit of 1 GB, or the limit of 2 GB for PAE. To find the most suitable value for this option, divide the desired address space in megabytes by four. In this example `512` for 2 GB.

Loader Tunables

Increases the kmem address space on all FreeBSD architectures. A test system with 1 GB of physical memory benefitted from adding these options to /boot/loader.conf and then restarting:

```
vm.kmem_size="330M"
vm.kmem_size_max="330M"
vfs.zfs.arc.max="40M"
vfs.zfs.vdev.cache.size="5M"
```

For a more detailed list of recommendations for ZFS-related tuning, see https://wiki.freebsd.org/ZFSTuningGuide.

23.7. Further Resources

- OpenZFS
- FreeBSD Wiki - ZFS Tuning
- Calomel Blog - ZFS Raidz Performance, Capacity and Integrity

23.8. ZFS Features and Terminology

More than a file system, ZFS is fundamentally different. ZFS combines the roles of file system and volume manager, enabling new storage devices to add to a live system and having the new space available on the existing file systems in that pool at once. By combining the traditionally separate roles, ZFS is able to overcome previous limitations that prevented RAID groups being able to grow. A *vdev* is a top level device in a pool and can be a simple disk or a RAID transformation such as a mirror or RAID-Z array. ZFS file systems (called *datasets*) each have access to the combined free space of the entire pool. Used blocks from the pool decrease the space available to each file system. This approach avoids the common pitfall with extensive partitioning where free space becomes fragmented across the partitions.

pool	A storage *pool* is the most basic building block of ZFS. A pool consists of one or more vdevs, the underlying devices that store the data. A pool is then used to create one or more file systems (datasets) or block devices (volumes). These datasets and volumes share the pool of remaining free space. Each pool is uniquely identified by a name and a GUID. The ZFS version number on the pool determines the features available.

vdev Types	A pool consists of one or more vdevs, which themselves are a single disk or a group of disks, transformed to a RAID. When using a lot of vdevs, ZFS spreads data across the vdevs to increase performance and maximize usable space. All vdevs must be at least 128 MB in size. • *Disk* - The most basic vdev type is a standard block device. This can be an entire disk (such as /dev/ada0 or /dev/da0) or a partition (/dev/ada0p3). On FreeBSD, there is no performance penalty for using a partition rather than the entire disk. This differs from recommendations made by the Solaris documentation. Using an entire disk as part of a bootable pool is strongly discouraged, as this may render the pool unbootable. Likewise, you should not use an entire disk as part of a mirror or RAID-Z vdev. Reliably determining the size of an unpartitioned disk at boot time is impossible and there's no place to put in boot code. • *File* - Regular files may make up ZFS pools, which is useful for testing and experimentation. Use the full path to the file as the device path in `zpool create`. • *Mirror* - When creating a mirror, specify the `mirror` keyword followed by the list of member devices for the mirror. A mirror consists of two or more devices, writing all data to all member devices. A mirror vdev will hold as much data as its smallest member. A mirror vdev can withstand the failure of all but one of its members without losing any data. To upgrade a regular single disk vdev to a mirror vdev at any time, use `zpool attach`. • *RAID-Z* - ZFS uses RAID-Z, a variation on standard RAID-5 that offers better distribution of parity and eliminates the "RAID-5 write hole" in which the data and parity information become inconsistent after an unexpected restart. ZFS supports three levels of RAID-Z which provide varying levels of redundancy in exchange for decreasing levels of usable storage. ZFS uses RAID-Z1 through RAID-Z3 based on the number of parity devices in the array and the number of disks which can fail before the pool stops being operational. In a RAID-Z1 configuration with four disks, each 1 TB, usable storage is 3 TB and the pool will still be able to operate in degraded mode with one faulted disk. If another disk goes offline before replacing and resilvering the faulted disk would result in losing all pool data. In a RAID-Z3 configuration with eight disks of 1 TB, the volume will provide 5 TB of usable space and still be able to operate with three faulted disks. Sun™ recommends no more than nine disks in a single vdev. If more disks make up the configuration, the recommendation is to divide them into separate vdevs and stripe the pool data across them. A configuration of two RAID-Z2 vdevs consisting of 8 disks each would create something like a RAID-60 array. A RAID-Z group's storage capacity is about the size of

Transaction Group (TXG)	Transaction Groups are the way ZFS groups blocks changes together and writes them to the pool. Transaction groups are the atomic unit that ZFS uses to ensure consistency. ZFS assigns each transaction group a unique 64-bit consecutive identifier. There can be up to three active transaction groups at a time, one in each of these three states: * *Open* - A new transaction group begins in the open state and accepts new writes. There is always a transaction group in the open state, but the transaction group may refuse new writes if it has reached a limit. Once the open transaction group has reached a limit, or reaching the `vfs.zfs.txg.timeout`, the transaction group advances to the next state. * *Quiescing* - A short state that allows any pending operations to finish without blocking the creation of a new open transaction group. Once all the transactions in the group have completed, the transaction group advances to the final state. * *Syncing* - Write all the data in the transaction group to stable storage. This process will in turn change other data, such as metadata and space maps, that ZFS will also write to stable storage. The process of syncing involves several passes. On the first and biggest, all the changed data blocks; next come the metadata, which may take several passes to complete. Since allocating space for the data blocks generates new metadata, the syncing state cannot finish until a pass completes that does not use any new space. The syncing state is also where *synctasks* complete. Synctasks are administrative operations such as creating or destroying snapshots and datasets that complete the uberblock change. Once the sync state completes the transaction group in the quiescing state advances to the syncing state. All administrative functions, such as `snapshot` write as part of the transaction group. ZFS adds a created synctask to the open transaction group, and that group advances as fast as possible to the syncing state to reduce the latency of administrative commands.
Adaptive Replacement Cache (ARC)	ZFS uses an Adaptive Replacement Cache (ARC), rather than a more traditional Least Recently Used (LRU) cache. An LRU cache is a simple list of items in the cache, sorted by how recently object was used, adding new items to the head of the list. When the cache is full, evicting items from the tail of the list makes room for more active objects. An ARC consists of four lists; the Most Recently Used (MRU) and Most Frequently Used (MFU) objects, plus a ghost list for each. These ghost lists track evicted objects to prevent adding them back to the cache. This increases the cache hit ratio by avoiding objects that have a history of occasional use. Another advantage of using both an MRU and MFU is that scanning an entire file system would evict all data from an MRU or LRU cache in favor of this freshly accessed content. With ZFS, there is also an MFU that tracks the most frequently used objects, and the cache of the most commonly accessed blocks remains.

L2ARC	L2ARC is the second level of the ZFS caching system. RAM stores the primary ARC. Since the amount of available RAM is often limited, ZFS can also use cache vdevs. Solid State Disks (SSDs) are often used as these cache devices due to their higher speed and lower latency compared to traditional spinning disks. L2ARC is entirely optional, but having one will increase read speeds for cached files on the SSD instead of having to read from the regular disks. L2ARC can also speed up deduplication because a deduplication table (DDT) that does not fit in RAM but does fit in the L2ARC will be much faster than a DDT that must read from disk. Limits on the data rate added to the cache devices prevents prematurely wearing out SSDs with extra writes. Until the cache is full (the first block evicted to make room), writes to the L2ARC limit to the sum of the write limit and the boost limit, and afterwards limit to the write limit. A pair of sysctl(8) values control these rate limits. `vfs.zfs.l2arc_write_max` controls the number of bytes written to the cache per second, while `vfs.zfs.l2arc_write_boost` adds to this limit during the "Turbo Warmup Phase" (Write Boost).
ZIL	ZIL accelerates synchronous transactions by using storage devices like SSDs that are faster than those used in the main storage pool. When an application requests a synchronous write (a guarantee that the data is stored to disk rather than merely cached for later writes), writing the data to the faster ZIL storage then later flushing it out to the regular disks greatly reduces latency and improves performance. Synchronous workloads like databases will profit from a ZIL alone. Regular asynchronous writes such as copying files will not use the ZIL at all.
Copy-On-Write	Unlike a traditional file system, ZFS writes a different block rather than overwriting the old data in place. When completing this write the metadata updates to point to the new location. When a shorn write (a system crash or power loss in the middle of writing a file) occurs, the entire original contents of the file are still available and ZFS discards the incomplete write. This also means that ZFS does not require a fsck(8) after an unexpected shutdown.
Dataset	*Dataset* is the generic term for a ZFS file system, volume, snapshot or clone. Each dataset has a unique name in the format *poolname/path@snapshot*. The root of the pool is a dataset as well. Child datasets have hierarchical names like directories. For example, *mypool/home*, the home dataset, is a child of *mypool* and inherits properties from it. Expand this further by creating *mypool/home/user*. This grandchild dataset will inherit properties from the parent and grandparent. Set properties on a child to override the defaults inherited from the parent and grandparent. Administration of datasets and their children can be delegated.
File system	A ZFS dataset is most often used as a file system. Like most other file systems, a ZFS file system mounts somewhere in the systems directory hierarchy and contains files and directories of its own with permissions, flags, and other metadata.
Volume	ZFS can also create volumes, which appear as disk devices. Volumes have a lot of the same features as datasets, including copy-on-write, snapshots, clones, and checksumming. Volumes can be useful for running other file system formats on top of ZFS, such as UFS virtualization, or exporting iSCSI extents.

Snapshot	The copy-on-write (COW) design of ZFS allows for nearly instantaneous, consistent snapshots with arbitrary names. After taking a snapshot of a dataset, or a recursive snapshot of a parent dataset that will include all child datasets, new data goes to new blocks, but without reclaiming the old blocks as free space. The snapshot contains the original file system version and the live file system contains any changes made since taking the snapshot using no other space. New data written to the live file system uses new blocks to store this data. The snapshot will grow as the blocks are no longer used in the live file system, but in the snapshot alone. Mount these snapshots read-only allows recovering of previous file versions. A rollback of a live file system to a specific snapshot is possible, undoing any changes that took place after taking the snapshot. Each block in the pool has a reference counter which keeps track of the snapshots, clones, datasets, or volumes use that block. As files and snapshots get deleted, the reference count decreases, reclaiming the free space when no longer referencing a block. Marking snapshots with a hold results in any attempt to destroy it will returns an `EBUSY` error. Each snapshot can have holds with a unique name each. The release command removes the hold so the snapshot can deleted. Snapshots, cloning, and rolling back works on volumes, but independently mounting does not.
Clone	Cloning a snapshot is also possible. A clone is a writable version of a snapshot, allowing the file system to fork as a new dataset. As with a snapshot, a clone initially consumes no new space. As new data written to a clone uses new blocks, the size of the clone grows. When blocks are overwritten in the cloned file system or volume, the reference count on the previous block decreases. Removing the snapshot upon which a clone bases is impossible because the clone depends on it. The snapshot is the parent, and the clone is the child. Clones can be *promoted*, reversing this dependency and making the clone the parent and the previous parent the child. This operation requires no new space. Since the amount of space used by the parent and child reverses, it may affect existing quotas and reservations.
Checksum	Every block is also checksummed. The checksum algorithm used is a per-dataset property, see `set`. The checksum of each block is transparently validated when read, allowing ZFS to detect silent corruption. If the data read does not match the expected checksum, ZFS will attempt to recover the data from any available redundancy, like mirrors or RAID-Z. Triggering a validation of all checksums with `scrub`. Checksum algorithms include: * `fletcher2` * `fletcher4` * `sha256` The `fletcher` algorithms are faster, but `sha256` is a strong cryptographic hash and has a much lower chance of collisions at the cost of some performance. Deactivating checksums is possible, but strongly discouraged.

Compression	Each dataset has a compression property, which defaults to off. Set this property to an available compression algorithm. This causes compression of all new data written to the dataset. Beyond a reduction in space used, read and write throughput often increases because fewer blocks need reading or writing. * *LZ4* - Added in ZFS pool version 5000 (feature flags), LZ4 is now the recommended compression algorithm. LZ4 works about 50% faster than LZJB when operating on compressible data, and is over three times faster when operating on uncompressible data. LZ4 also decompresses about 80% faster than LZJB. On modern CPUs, LZ4 can often compress at over 500 MB/s, and decompress at over 1.5 GB/s (per single CPU core). * *LZJB* - The default compression algorithm. Created by Jeff Bonwick (one of the original creators of ZFS). LZJB offers good compression with less CPU overhead compared to GZIP. In the future, the default compression algorithm will change to LZ4. * *GZIP* - A popular stream compression algorithm available in ZFS. One of the main advantages of using GZIP is its configurable level of compression. When setting the `compress` property, the administrator can choose the level of compression, ranging from `gzip1`, the lowest level of compression, to `gzip9`, the highest level of compression. This gives the administrator control over how much CPU time to trade for saved disk space. * *ZLE* - Zero Length Encoding is a special compression algorithm that compresses continuous runs of zeros alone. This compression algorithm is useful when the dataset contains large blocks of zeros.
Copies	When set to a value greater than 1, the `copies` property instructs ZFS to maintain copies of each block in the crossref:zfs[zfs-term-filesystem,file system] or volume. Setting this property on important datasets provides added redundancy from which to recover a block that does not match its checksum. In pools without redundancy, the copies feature is the single form of redundancy. The copies feature can recover from a single bad sector or other forms of minor corruption, but it does not protect the pool from the loss of an entire disk.

Deduplication	Checksums make it possible to detect duplicate blocks when writing data. With deduplication, the reference count of an existing, identical block increases, saving storage space. ZFS keeps a deduplication table (DDT) in memory to detect duplicate blocks. The table contains a list of unique checksums, the location of those blocks, and a reference count. When writing new data, ZFS calculates checksums and compares them to the list. When finding a match it uses the existing block. Using the SHA256 checksum algorithm with deduplication provides a secure cryptographic hash. Deduplication is tunable. If `dedup` is `on`, then a matching checksum means that the data is identical. Setting `dedup` to `verify`, ZFS performs a byte-for-byte check on the data ensuring they are actually identical. If the data is not identical, ZFS will note the hash collision and store the two blocks separately. As the DDT must store the hash of each unique block, it consumes a large amount of memory. A general rule of thumb is 5-6 GB of ram per 1 TB of deduplicated data). In situations not practical to have enough RAM to keep the entire DDT in memory, performance will suffer greatly as the DDT must read from disk before writing each new block. Deduplication can use L2ARC to store the DDT, providing a middle ground between fast system memory and slower disks. Consider using compression instead, which often provides nearly as much space savings without the increased memory.
Scrub	Instead of a consistency check like fsck(8), ZFS has `scrub`. `scrub` reads all data blocks stored on the pool and verifies their checksums against the known good checksums stored in the metadata. A periodic check of all the data stored on the pool ensures the recovery of any corrupted blocks before needing them. A scrub is not required after an unclean shutdown, but good practice is at least once every three months. ZFS verifies the checksum of each block during normal use, but a scrub makes certain to check even infrequently used blocks for silent corruption. ZFS improves data security in archival storage situations. Adjust the relative priority of `scrub` with `vfs.zfs.scrub_delay` to prevent the scrub from degrading the performance of other workloads on the pool.
Dataset Quota	ZFS provides fast and accurate dataset, user, and group space accounting as well as quotas and space reservations. This gives the administrator fine grained control over space allocation and allows reserving space for critical file systems. ZFS supports different types of quotas: the dataset quota, the reference quota (refquota), the user quota, and the group quota. Quotas limit the total size of a dataset and its descendants, including snapshots of the dataset, child datasets, and the snapshots of those datasets. Volumes do not support quotas, as the `volsize` property acts as an implicit quota.
Reference Quota	A reference quota limits the amount of space a dataset can consume by enforcing a hard limit. This hard limit includes space referenced by the dataset alone and does not include space used by descendants, such as file systems or snapshots.
User Quota	User quotas are useful to limit the amount of space used by the specified user.

Group Quota	The group quota limits the amount of space that a specified group can consume.
Dataset Reservation	The `reservation` property makes it possible to guarantee an amount of space for a specific dataset and its descendants. This means that setting a 10 GB reservation on storage/home/bob prevents other datasets from using up all free space, reserving at least 10 GB of space for this dataset. Unlike a regular `refreservation`, space used by snapshots and descendants is not counted against the reservation. For example, if taking a snapshot of storage/home/bob, enough disk space other than the `refreservation` amount must exist for the operation to succeed. Descendants of the main data set are not counted in the `refreservation` amount and so do not encroach on the space set. Reservations of any sort are useful in situations such as planning and testing the suitability of disk space allocation in a new system, or ensuring that enough space is available on file systems for audio logs or system recovery procedures and files.
Reference Reservation	The `refreservation` property makes it possible to guarantee an amount of space for the use of a specific dataset *excluding* its descendants. This means that setting a 10 GB reservation on storage/home/bob, and another dataset tries to use the free space, reserving at least 10 GB of space for this dataset. In contrast to a regular reservation, space used by snapshots and descendant datasets is not counted against the reservation. For example, if taking a snapshot of storage/home/bob, enough disk space other than the `refreservation` amount must exist for the operation to succeed. Descendants of the main data set are not counted in the `refreservation` amount and so do not encroach on the space set.
Resilver	When replacing a failed disk, ZFS must fill the new disk with the lost data. *Resilvering* is the process of using the parity information distributed across the remaining drives to calculate and write the missing data to the new drive.
Online	A pool or vdev in the `Online` state has its member devices connected and fully operational. Individual devices in the `Online` state are functioning.
Offline	The administrator puts individual devices in an `Offline` state if enough redundancy exists to avoid putting the pool or vdev into a Faulted state. An administrator may choose to offline a disk in preparation for replacing it, or to make it easier to identify.
Degraded	A pool or vdev in the `Degraded` state has one or more disks that disappeared or failed. The pool is still usable, but if other devices fail, the pool may become unrecoverable. Reconnecting the missing devices or replacing the failed disks will return the pool to an Online state after the reconnected or new device has completed the Resilver process.
Faulted	A pool or vdev in the `Faulted` state is no longer operational. Accessing the data is no longer possible. A pool or vdev enters the `Faulted` state when the number of missing or failed devices exceeds the level of redundancy in the vdev. If reconnecting missing devices the pool will return to an Online state. Insufficient redundancy to compensate for the number of failed disks loses the pool contents and requires restoring from backups.

Chapter 24. Other File Systems

24.1. Synopsis

File systems are a fundamental component of any operating system. They enable users to store, manage, and access data, making storage devices like hard drives, flash drives, and USB storage devices practical for everyday use. Different operating systems use different file systems natively.

FreeBSD has traditionally used the Unix File System (UFS), with the modernized UFS2 as its primary native file system. FreeBSD also uses the Z File System (ZFS), known for its advanced features, robustness, and reliability. See The Z File System (ZFS) for more details.

In addition to its native file systems, FreeBSD supports a broad range of file systems from other operating systems. Support for these file systems varies, some require loading kernel modules, while others need additional userland tools.

Before reading this chapter:

- Be familiar with UNIX® concepts and FreeBSD basics.
- Feel comfortable installing software in FreeBSD.
- Have some understanding of disks, storage devices, and FreeBSD's naming conventions for devices.

Read this chapter to:

- Understand the differences between native and supported file systems.
- Know which file systems are supported by FreeBSD and the level of support available.
- Learn how to enable, configure, access, and work with non-native file systems.

24.2. Linux® File Systems

FreeBSD provides built-in support for several Linux® file systems. This section demonstrates how to load support for and how to mount the supported Linux® file systems.

24.2.1. Extended File System (EXT)

Kernel support for Extended File System (EXT) file systems has been available since FreeBSD 2.2. The ext2fs(4) driver allows the FreeBSD kernel to both read and write to ext2, ext3, and ext4 file systems.

Journalling and encryption are not supported yet.

To access an ext file system, mount the ext volume by specifying its FreeBSD partition name and an existing mount point. This example mounts /dev/ada1s1 on /mnt:

```
# mount -t ext2fs /dev/ada1s1 /mnt
```

24.3. Windows® File Systems

FreeBSD supports FAT, exFAT, and NTFS file systems, enabling access to Windows-formatted storage.

24.3.1. FAT File System

The FAT file system is a simple and robust file system. Although it lacks the performance, reliability, and scalability of modern alternatives, its availability on many different operating systems makes it a common choice for data exchange between devices.

To access a FAT file system, mount the FAT volume by specifying its FreeBSD partition name and an existing mount point. This example mounts /dev/ada0s1 on /mnt:

```
# mount -t msdosfs /dev/ada0s1 /mnt
```

24.3.2. exFAT File System

exFAT (Extended File Allocation Table) is a lightweight file system optimized for flash storage devices, such as USB drives and SD cards. It supports large file sizes and is widely used across various platforms, making it ideal for external storage.

To use exFAT on FreeBSD, install the filesystems/exfat package, load the FUSE kernel module, and mount the file system as shown below:

Install the exFAT package:

```
# pkg install fusefs-exfat
```

Before using a FUSE file system, load the fusefs(4) kernel module:

```
# kldload fusefs
```

Use sysrc(8) to load the module at startup:

```
# sysrc kld_list+=fusefs
```

Mount the exFAT volume by specifying its FreeBSD partition name and an existing mount point. This example mounts /dev/ada0s1 on /mnt:

```
# mount.exfat /dev/ada0s1 /mnt
```

24.3.3. NTFS File System

NTFS is a robust file system developed by Microsoft® and commonly used in Windows operating systems. FreeBSD provides full read and write support for NTFS through the filesystems/ntfs package, making it easy to access and modify NTFS-formatted storage devices.

To use NTFS on FreeBSD, install the filesystems/ntfs package, load the FUSE kernel module, and mount the file system as shown below:

Install the NTFS package:

```
# pkg install fusefs-ntfs
```

Before using a FUSE file system, load the fusefs(4) kernel module:

```
# kldload fusefs
```

Use sysrc(8) to load the module at startup:

```
# sysrc kld_list+=fusefs
```

Mount the NTFS volume by specifying its FreeBSD partition name and an existing mount point. This example mounts /dev/ada0s1 on /mnt:

```
# ntfs-3g /dev/ada0s1 /mnt
```

24.4. MacOS® File Systems

FreeBSD provides support for MacOS® file systems, including HFS/HFS+, allowing access to storage devices formatted for Apple® systems.

24.4.1. HFS/HFS+ File System

HFS/HFS+ was the primary file system for MacOS prior to APFS, commonly used on older Mac devices and external drives. FreeBSD provides read-only support for HFS/HFS+ through the filesystems/hfsfuse package.

To use HFS/HFS+ on FreeBSD, install the filesystems/hfsfuse package, load the FUSE kernel module, and mount the file system as shown below:

Install the HFS/HFS+ package:

```
# pkg install fusefs-hfsfuse
```

Before using a FUSE file system, load the fusefs(4) kernel module:

```
# kldload fusefs
```

Use sysrc(8) to load the module at startup:

```
# sysrc kld_list+=fusefs
```

Mount the HFS/HFS+ volume by specifying its FreeBSD partition name and an existing mount point. This example mounts /dev/ada0s1 on /mnt:

```
# hfsfuse /dev/ada0s1 /mnt
```

Chapter 25. Virtualization

25.1. Synopsis

Virtualization software allows multiple operating systems to run simultaneously on the same computer. Such software systems for PCs often involve a host operating system which runs the virtualization software and supports any number of guest operating systems.

Read this chapter to learn:

- The difference between a host operating system and a guest operating system.
- How to install FreeBSD on the following virtualization platforms:
    - Parallels Desktop(Apple® macOS®)
    - VMware Fusion(Apple® macOS®)
    - VirtualBox™(Microsoft® Windows®, Intel®-based Apple® macOS®, Linux)
    - QEMU(FreeBSD)
    - bhyve(FreeBSD)
- How to tune a FreeBSD system for best performance under virtualization.

Before reading this chapter:

- Understand the basics of UNIX® and FreeBSD.
- Know how to install FreeBSD.
- Know how to set up a network connection.
- Know how to install additional third-party software.

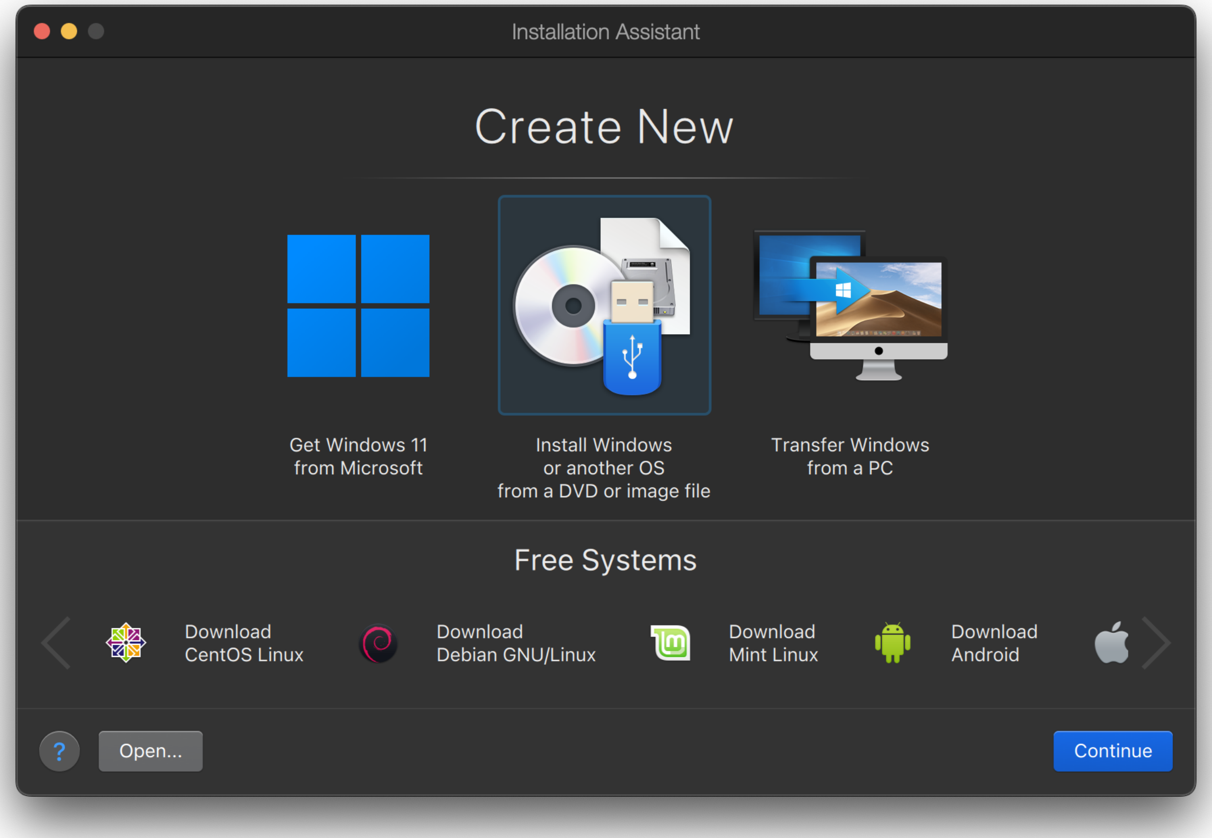
25.2. FreeBSD as a Guest on Parallels Desktop for macOS®

Parallels Desktop for Mac® is a commercial software product available for Apple® Mac® computers running macOS® 10.14.6 or higher. FreeBSD is a fully supported guest operating system. Once Parallels has been installed on macOS®, the user must configure a virtual machine and then install the desired guest operating system.

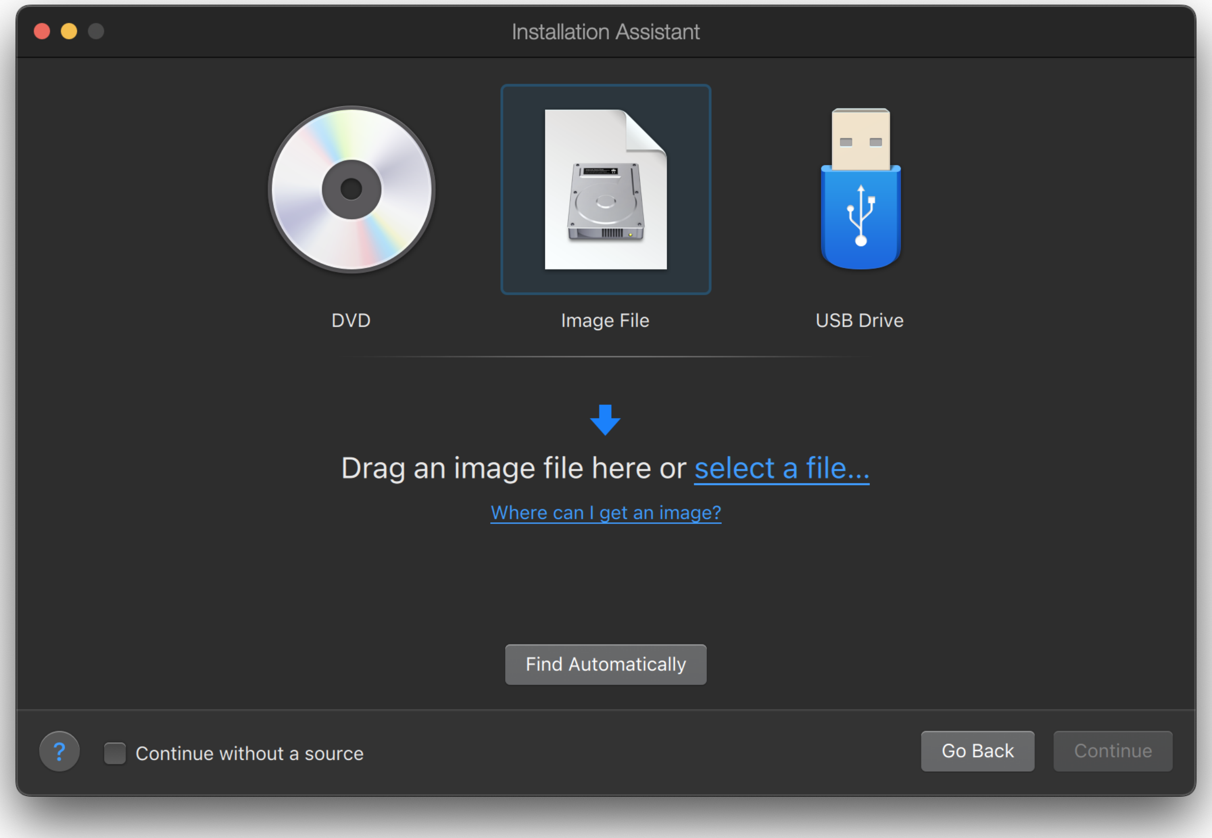
25.2.1. Installing FreeBSD on Parallels Desktop on Mac®

The first step in installing FreeBSD on Parallels is to create a new virtual machine for installing FreeBSD.

Choose **Install Windows or another OS from a DVD or image file** and proceed.

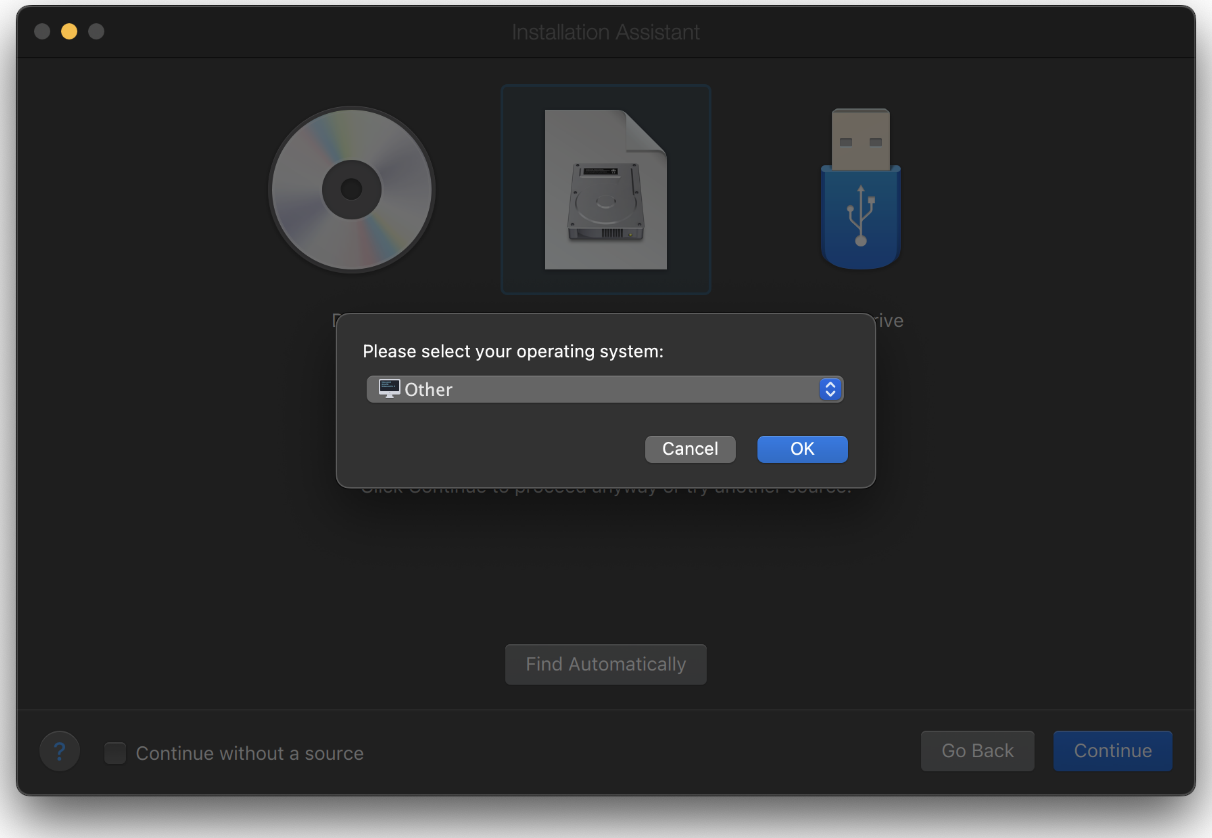

Installation Assistant
Create New
Get Windows 11 from Microsoft
Install Windows or another OS from a DVD or image file
Transfer Windows from a PC
Free Systems
Download CentOS Linux
Download Debian GNU/Linux
Download Mint Linux
Download Android
?
Open...
Continue

Select the FreeBSD image file.


Installation Assistant
DVD
Image File
USB Drive
Drag an image file here or select a file...
Where can I get an image?
Find Automatically
?
☐ Continue without a source
Go Back
Continue

Choose **Other as operating system**.

Choosing FreeBSD will cause boot error on startup.

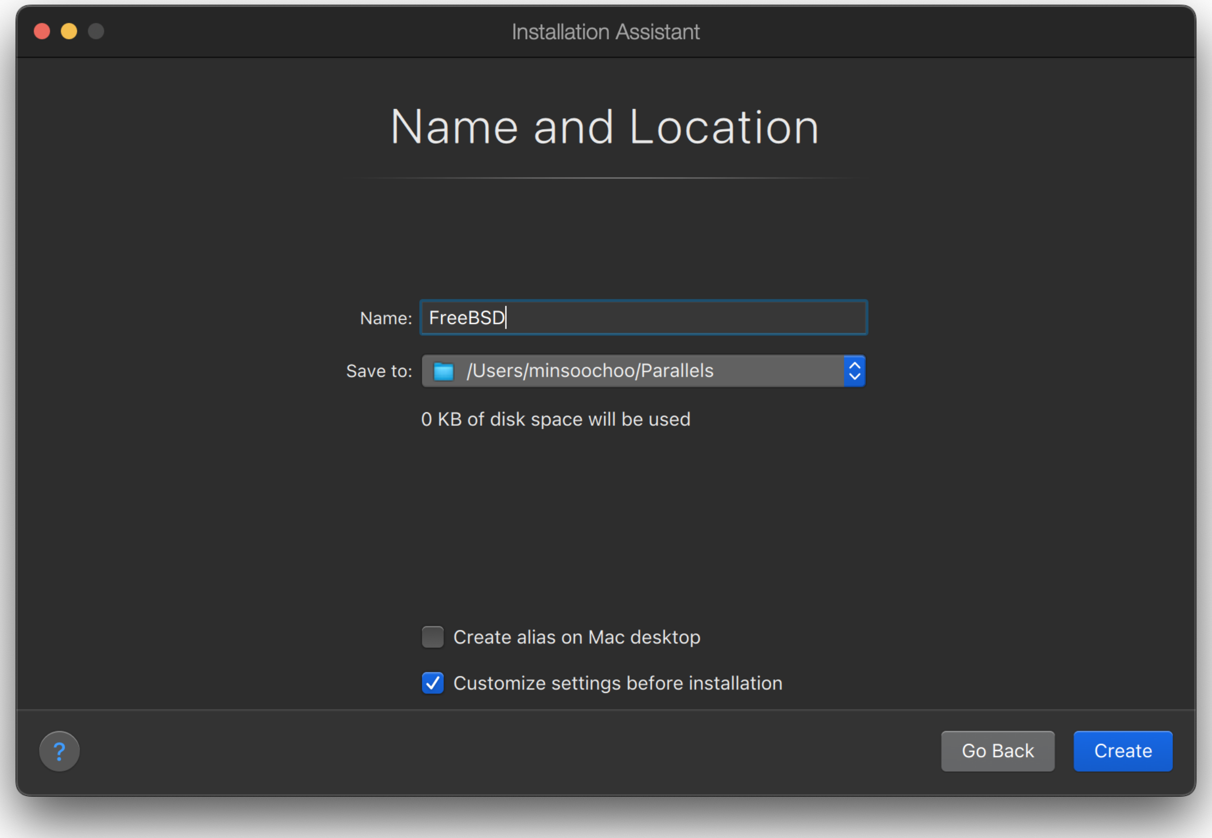


Please select your operating system:

Other

Installation Assistant
Cancel
OK
Find Automatically
Continue without a source
Go Back
Continue

Name the virtual machine and check **Customize settings before installation**


Installation Assistant
Name and Location

Name: FreeBSD

Save to: /Users/minsoochoo/Parallels

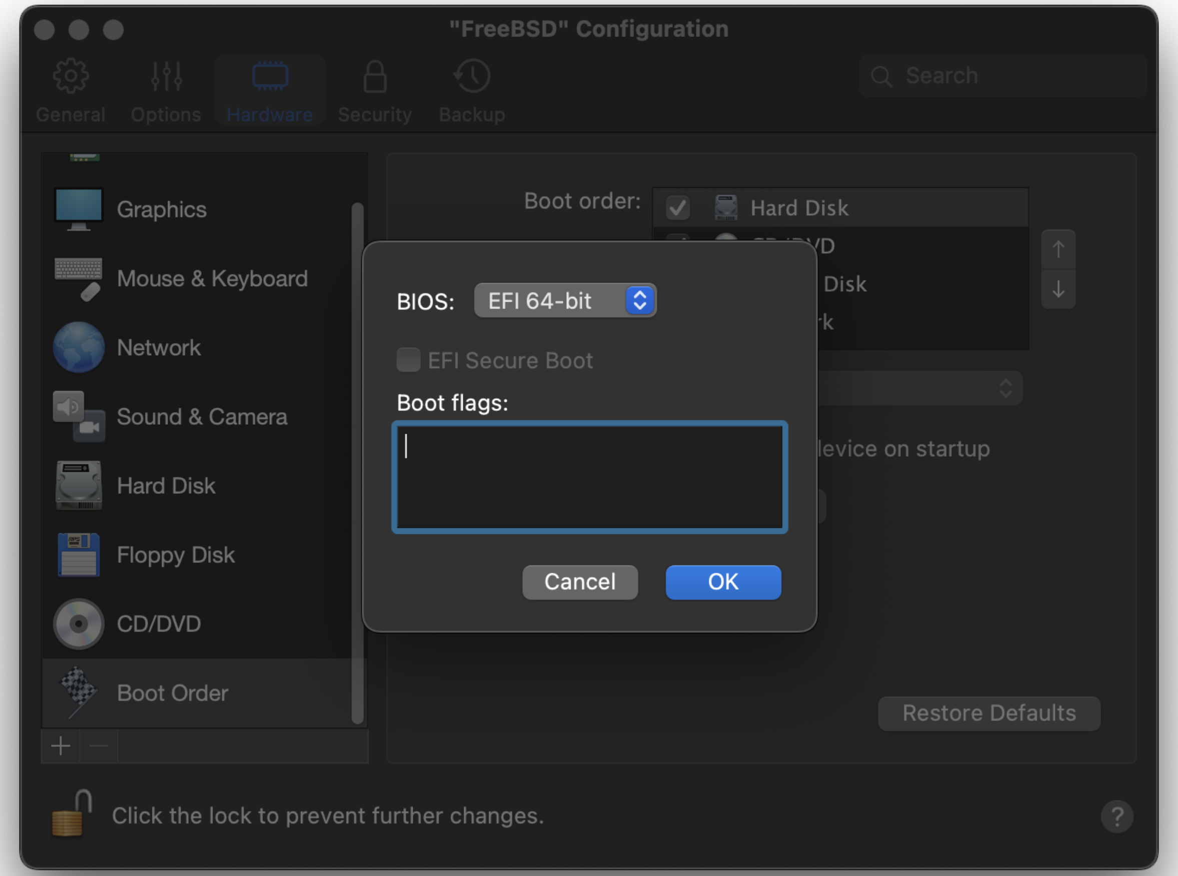
0 KB of disk space will be used

☐ Create alias on Mac desktop

☑ Customize settings before installation

?
Go Back
Create

When the configuration window pops up, go to **Hardware** tab, choose **Boot order**, and click **Advanced**. Then, choose **EFI 64-bit** as **BIOS**.



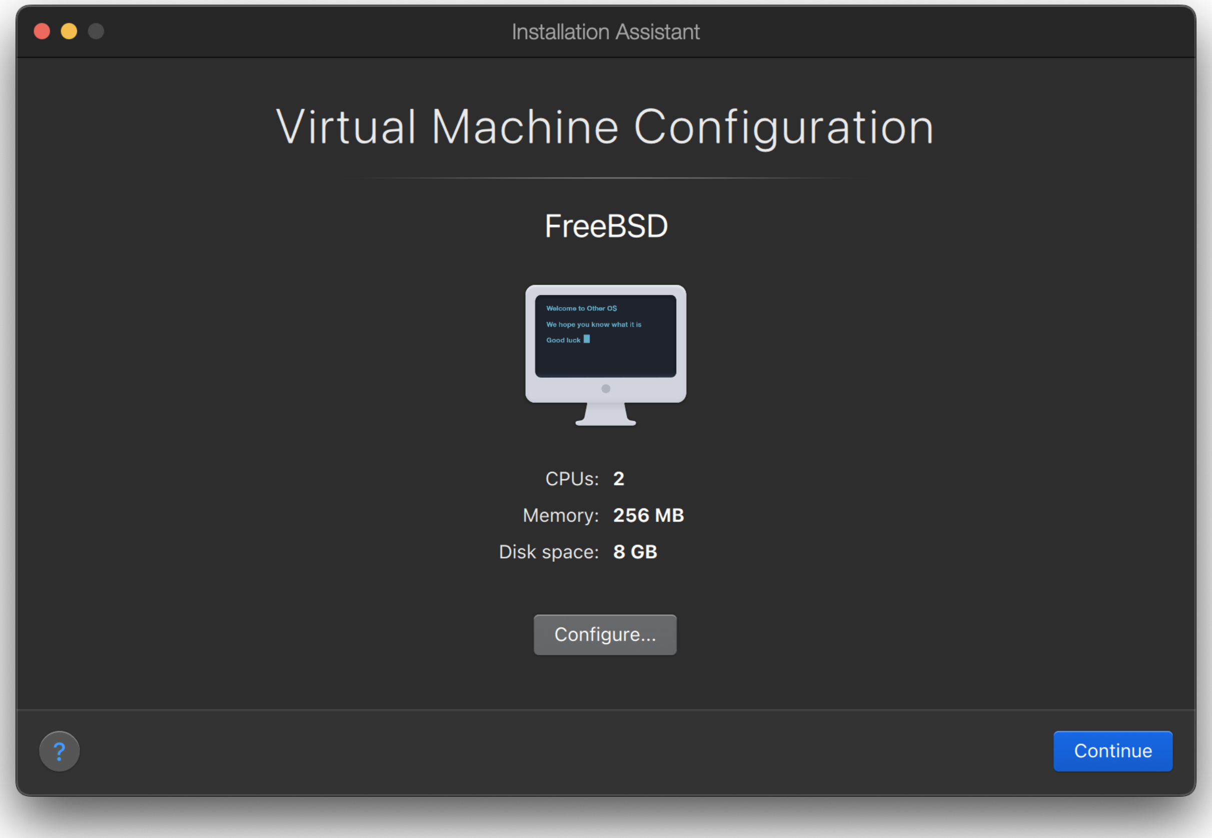
"FreeBSD" Configuration
General
Options
Hardware
Security
Backup
Search
Graphics
Mouse & Keyboard
Network
Sound & Camera
Hard Disk
Floppy Disk
CD/DVD
Boot Order
Boot order:
Hard Disk
Restore Defaults
Cancel
OK
Click the lock to prevent further changes.

BIOS: EFI 64-bit

☐ EFI Secure Boot

Boot flags:

Click **OK**, close the configuration window, and click **Continue**.



Installation Assistant

Virtual Machine Configuration

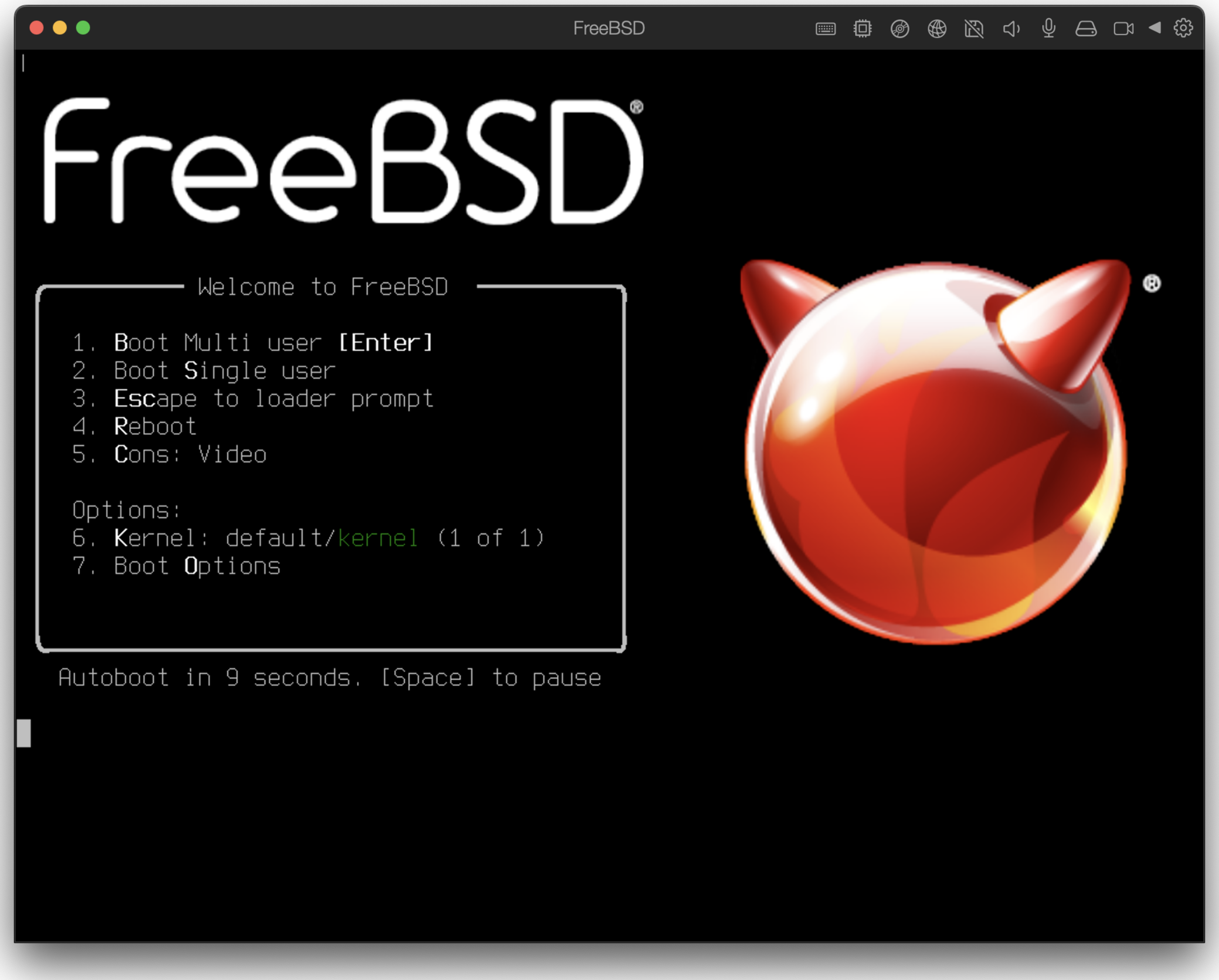
FreeBSD

Welcome to Other OS
We hope you know what it is
Good luck

CPUs: 2
Memory: 256 MB
Disk space: 8 GB

Configure...
?
Continue

The virtual machine will automatically boot. Install FreeBSD following the general steps.



```
Welcome to FreeBSD

1. Boot Multi user [Enter]
2. Boot Single user
3. Escape to loader prompt
4. Reboot
5. Cons: Video

Options:
6. Kernel: default/kernel (1 of 1)
7. Boot Options

Autoboot in 9 seconds. [Space] to pause
```

25.2.2. Configuring FreeBSD on Parallels

After FreeBSD has been successfully installed on macOS® X with Parallels, there are a number of configuration steps that can be taken to optimize the system for virtualized operation.

1. Set Boot Loader Variables

   The most important step is to reduce the `kern.hz` tunable to reduce the CPU utilization of FreeBSD under the Parallels environment. This is accomplished by adding the following line to /boot/loader.conf:

   ```
   kern.hz=100
   ```

   Without this setting, an idle FreeBSD Parallels guest will use roughly 15% of the CPU of a single processor iMac®. After this change the usage will be closer to 5%.

   If installing FreeBSD 14.0 or later, and CPU utilization is still high, add the following additional line to /boot/loader.conf:

   ```
   debug.acpi.disabled="ged"
   ```

2. Create a New Kernel Configuration File

   All SCSI, FireWire, and USB device drivers can be removed from a custom kernel configuration file. Parallels provides a virtual network adapter used by the ed(4) driver, so all network devices except for ed(4) and miibus(4) can be removed from the kernel.

3. Configure Networking

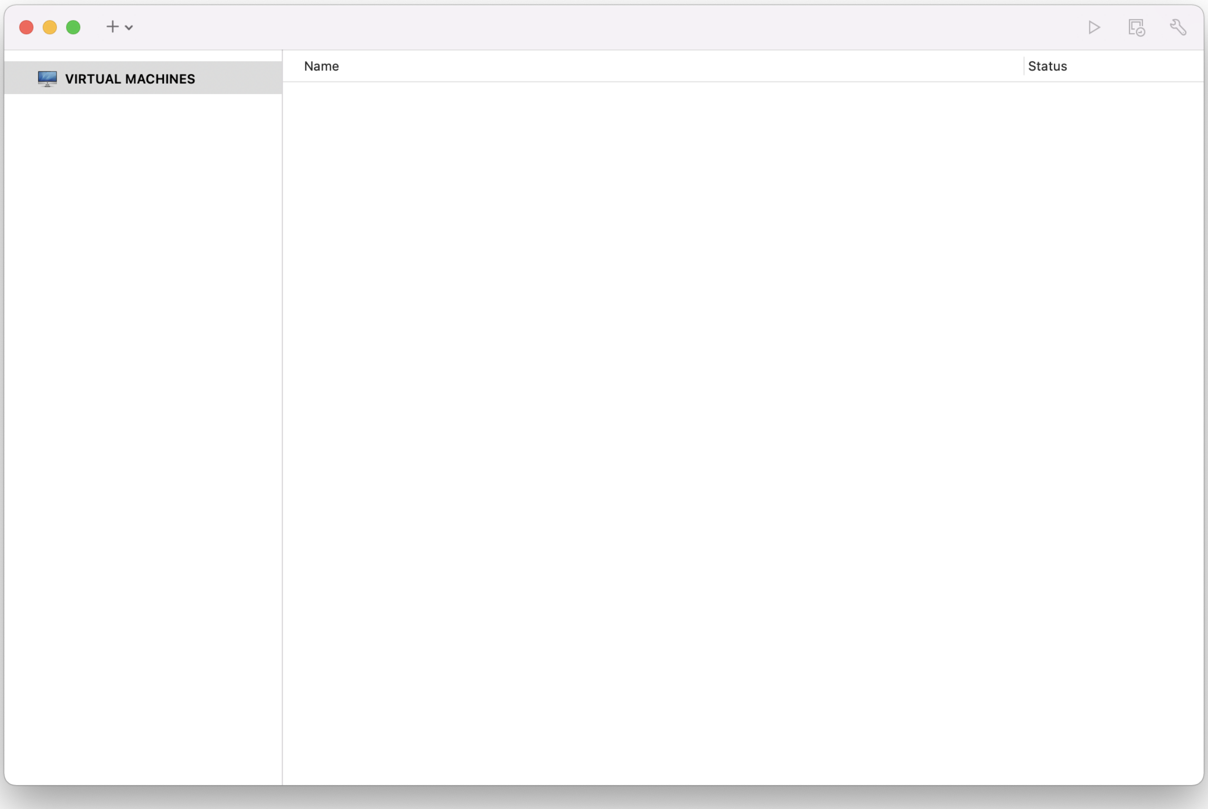
   The most basic networking setup uses DHCP to connect the virtual machine to the same local area network as the host Mac®. This can be accomplished by adding `ifconfig_ed0="DHCP"` to /etc/rc.conf. More advanced networking setups are described in Advanced Networking.

25.3. FreeBSD as a Guest on VMware Fusion for macOS®

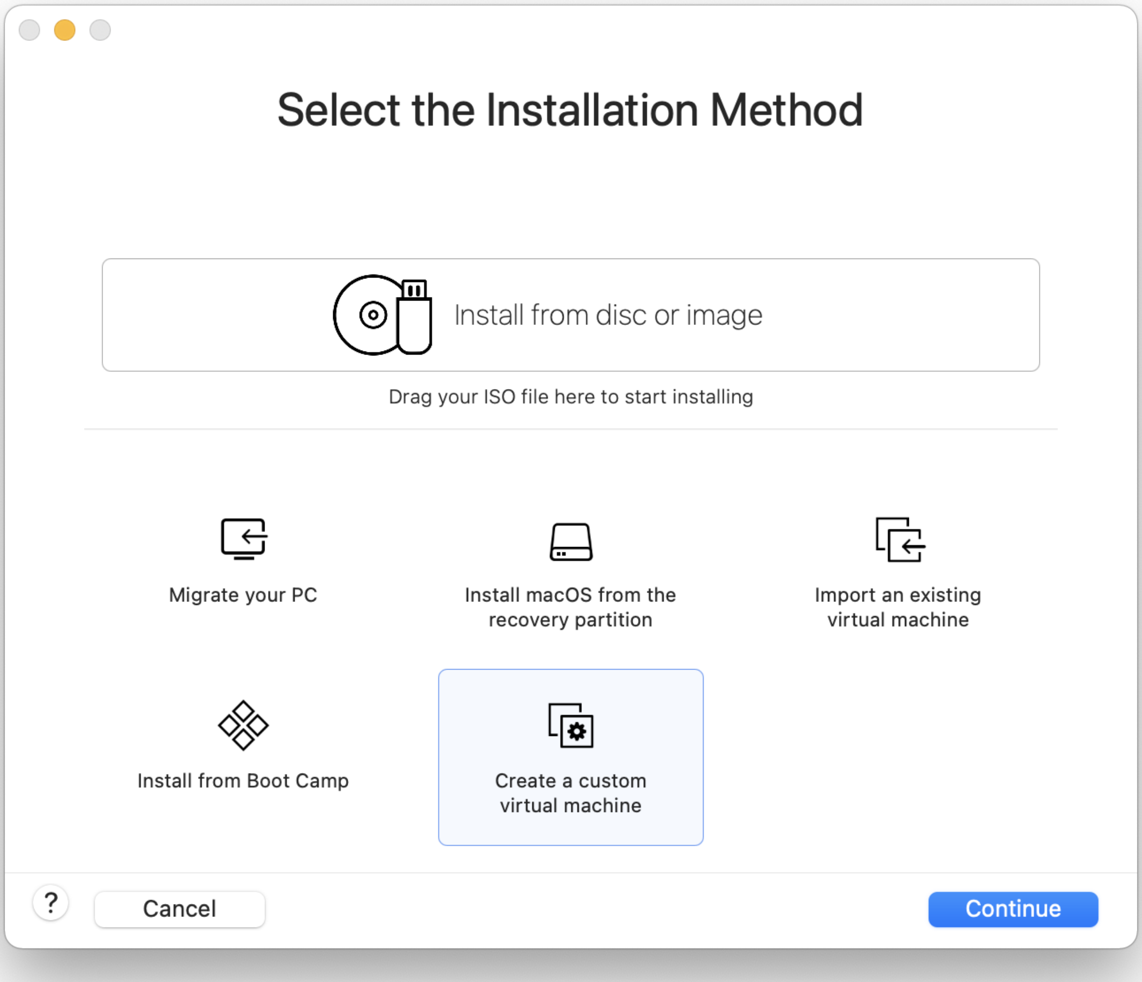
VMware Fusion for Mac® is a commercial software product available for Apple® Mac® computers running macOS® 12 or higher. FreeBSD is a fully supported guest operating system. Once VMware Fusion has been installed on macOS®, the user can configure a virtual machine and then install the desired guest operating system.

25.3.1. Installing FreeBSD on VMware Fusion

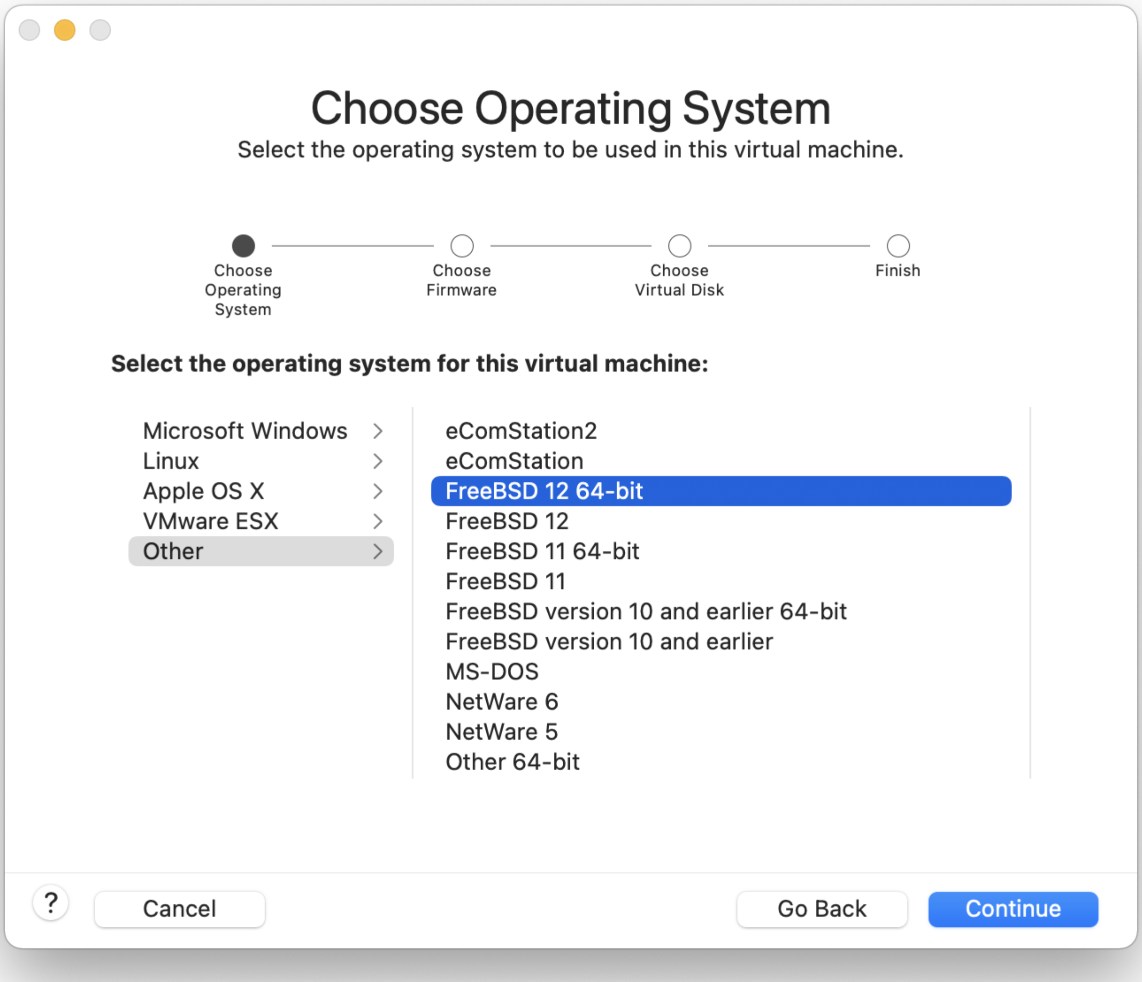
The first step is to start VMware Fusion which will load the Virtual Machine Library. Click + → New to create the virtual machine:



This will load the New Virtual Machine Assistant. Choose Create a custom virtual machine and click Continue to proceed:


Select the Installation Method
Install from disc or image
Drag your ISO file here to start installing
Migrate your PC
Install macOS from the recovery partition
Import an existing virtual machine
Install from Boot Camp
Create a custom virtual machine
?
Cancel
Continue

Select Other as the Operating System and either FreeBSD X or FreeBSD X 64-bit, as the **Version** when prompted:



Choose Operating System

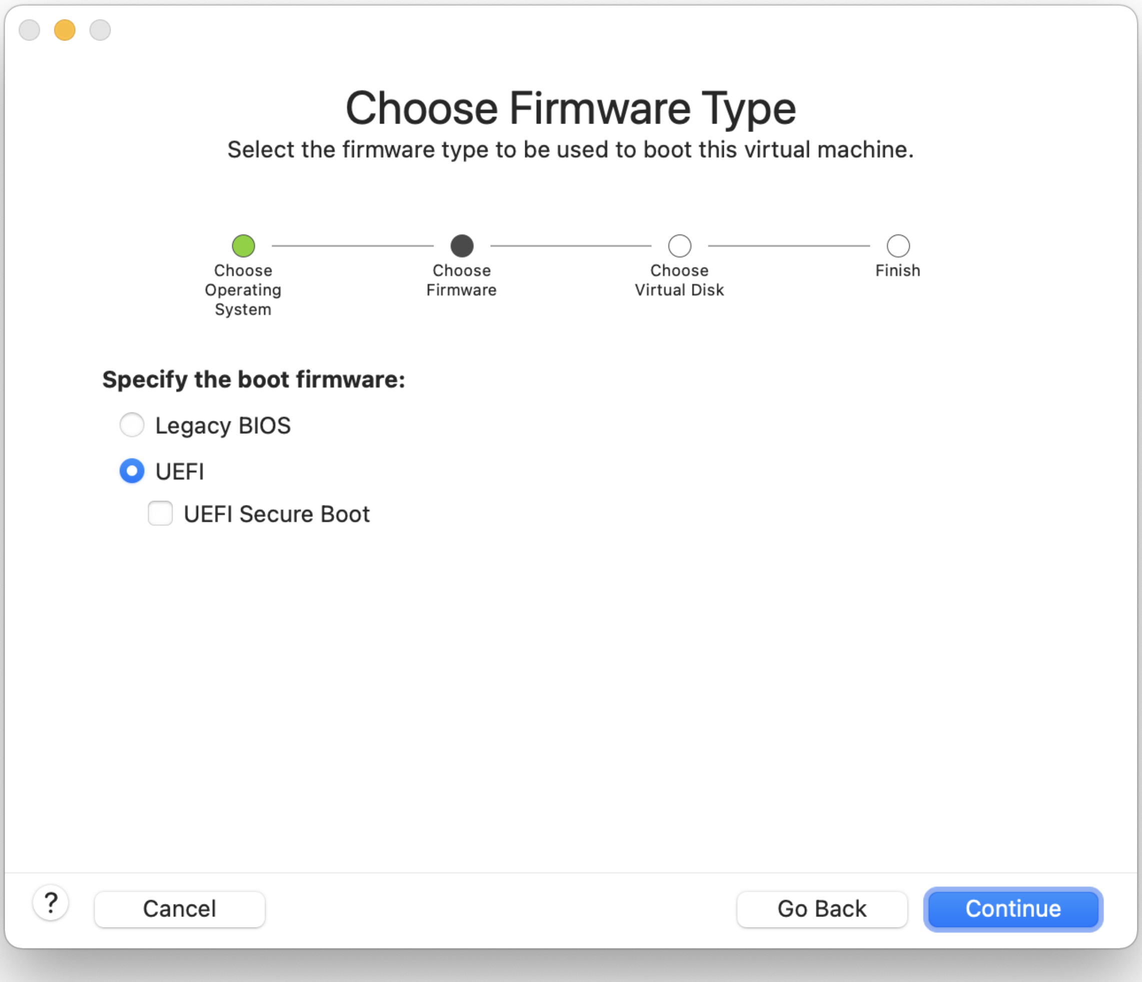
Select the operating system to be used in this virtual machine.

Choose Operating System
Choose Firmware
Choose Virtual Disk
Finish

Select the operating system for this virtual machine:

Microsoft Windows
Linux
Apple OS X
VMware ESX
Other
eComStation2
eComStation
FreeBSD 12 64-bit
FreeBSD 12
FreeBSD 11 64-bit
FreeBSD 11
FreeBSD version 10 and earlier 64-bit
FreeBSD version 10 and earlier
MS-DOS
NetWare 6
NetWare 5
Other 64-bit
?
Cancel
Go Back
Continue

Choose the firmware(UEFI is recommended):



Choose Firmware Type
Select the firmware type to be used to boot this virtual machine.
Choose Operating System
Choose Firmware
Choose Virtual Disk
Finish

Specify the boot firmware:

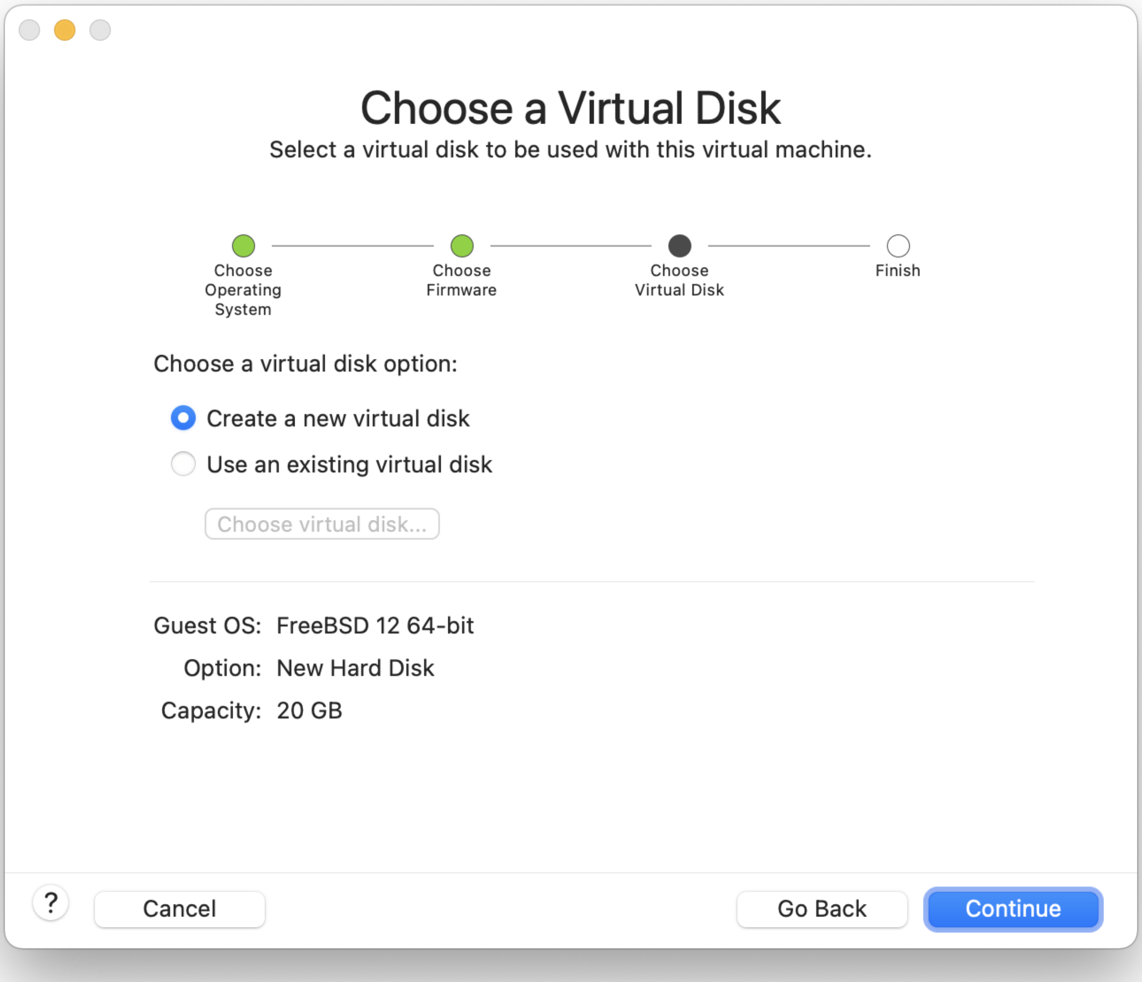
○ Legacy BIOS

● UEFI

  ☐ UEFI Secure Boot

?
Cancel
Go Back
Continue

Choose Create a new virtual disk and click Continue:



Choose a Virtual Disk

Select a virtual disk to be used with this virtual machine.

Choose Operating System
Choose Firmware
Choose Virtual Disk
Finish

Choose a virtual disk option:

● Create a new virtual disk

○ Use an existing virtual disk

Choose virtual disk...

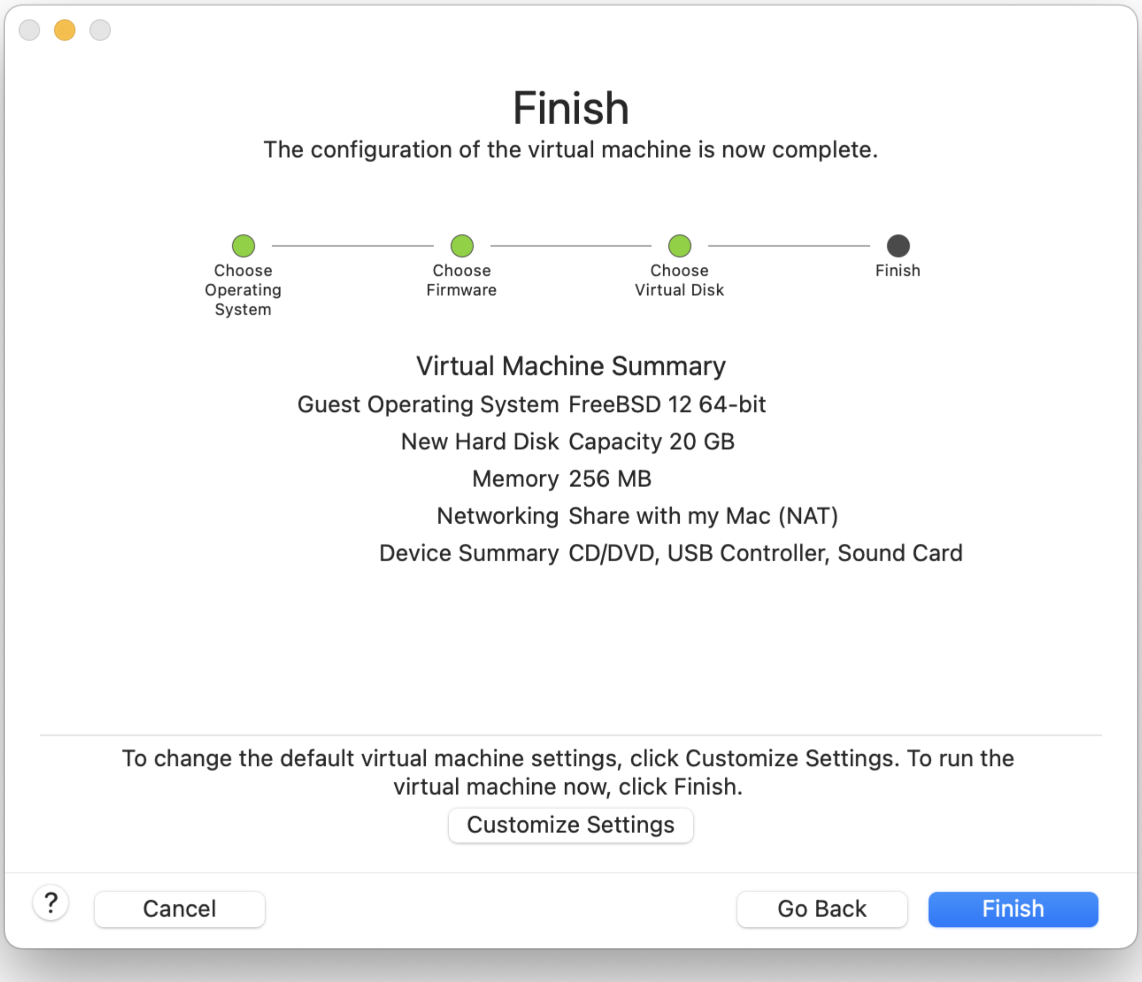
Guest OS: FreeBSD 12 64-bit

Option: New Hard Disk

Capacity: 20 GB

?
Cancel
Go Back
Continue

Check the configuration and click Finish:



Finish
The configuration of the virtual machine is now complete.
Choose Operating System
Choose Firmware
Choose Virtual Disk
Finish

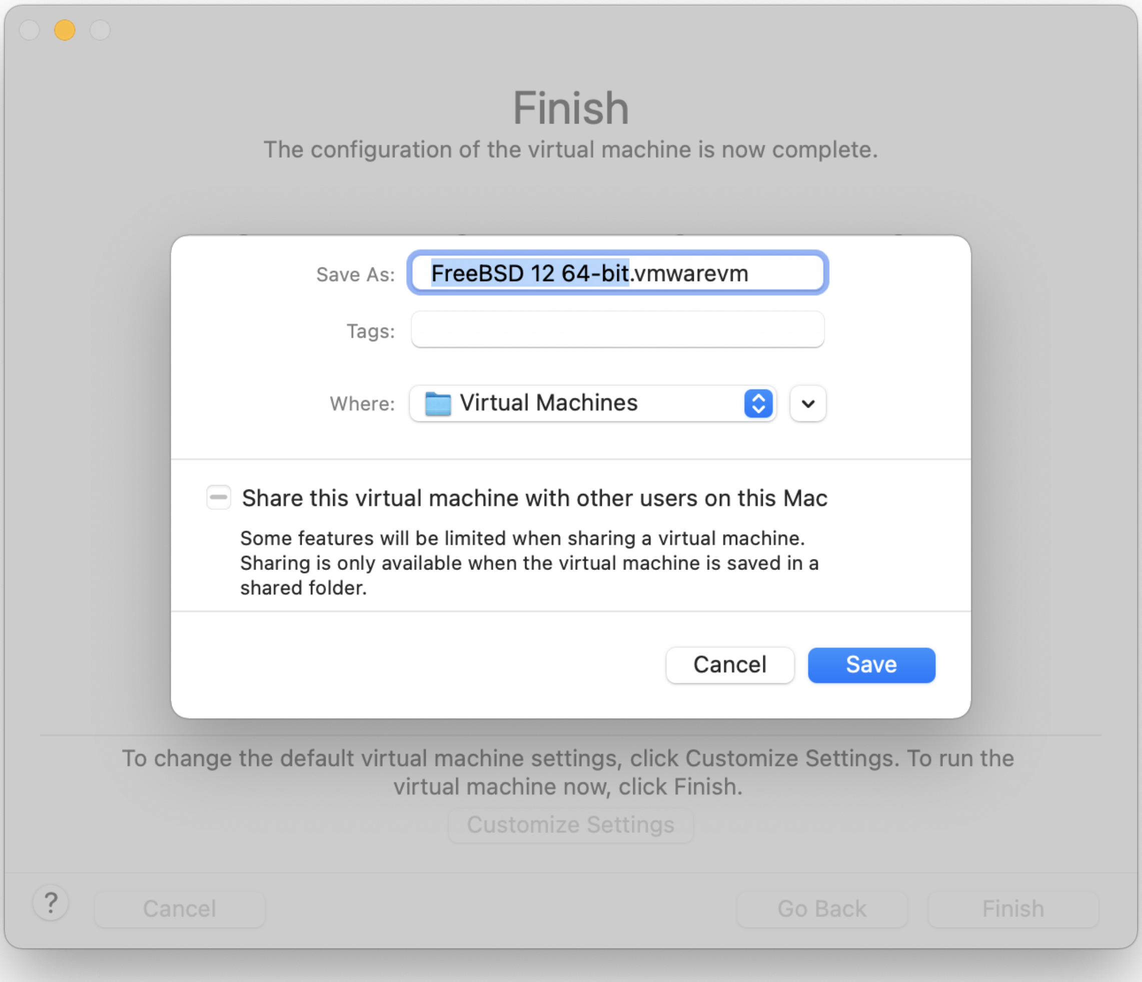
Virtual Machine Summary

Guest Operating System	FreeBSD 12 64-bit
New Hard Disk	Capacity 20 GB
Memory	256 MB
Networking	Share with my Mac (NAT)
Device Summary	CD/DVD, USB Controller, Sound Card

To change the default virtual machine settings, click Customize Settings. To run the virtual machine now, click Finish.

Customize Settings
?
Cancel
Go Back
Finish

Choose the name of the virtual machine and the directory where it should be saved:



Finish

The configuration of the virtual machine is now complete.

Save As: FreeBSD 12 64-bit.vmwarevm

Tags:

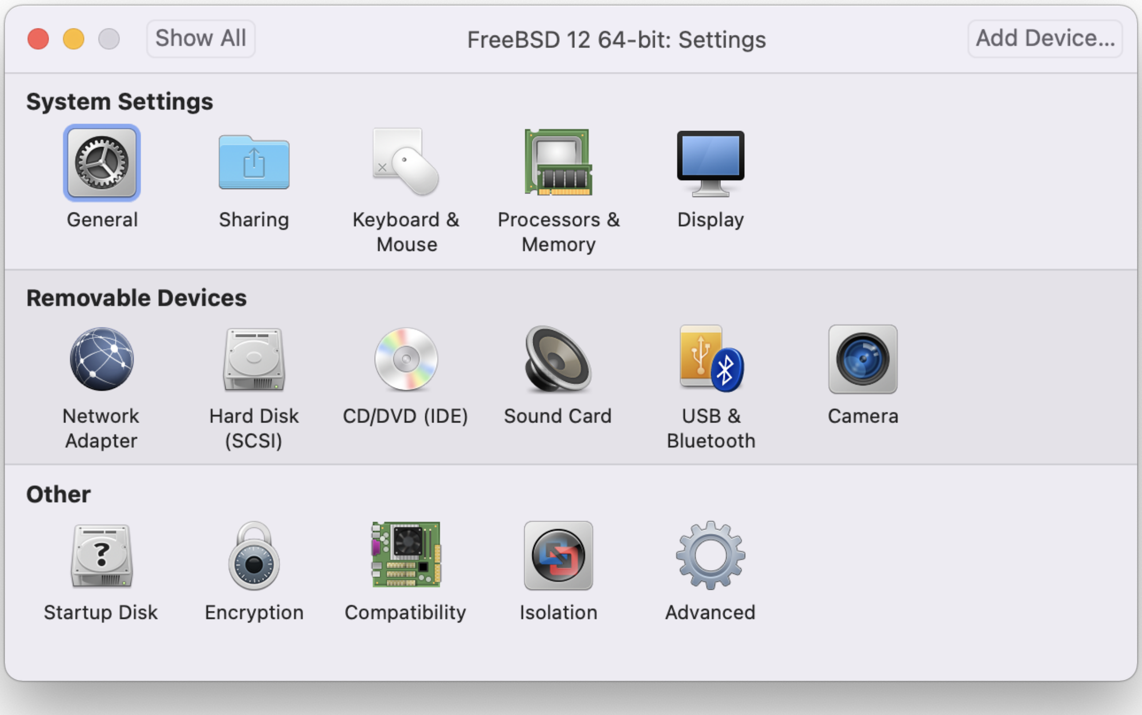
Where: Virtual Machines

☐ Share this virtual machine with other users on this Mac

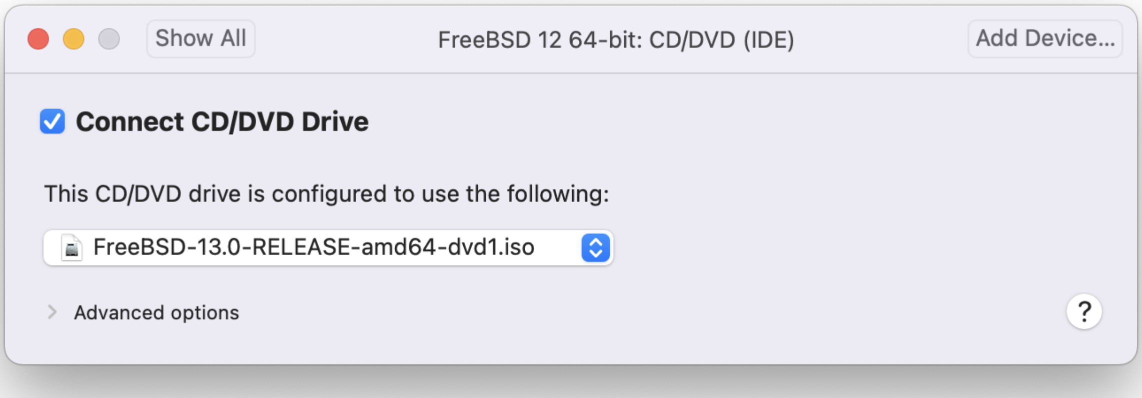
Some features will be limited when sharing a virtual machine. Sharing is only available when the virtual machine is saved in a shared folder.

To change the default virtual machine settings, click Customize Settings. To run the virtual machine now, click Finish.

Press command+E to open virtual machine settings and click CD/DVD:


Show All
FreeBSD 12 64-bit: Settings
Add Device...
System Settings
General
Sharing
Keyboard & Mouse
Processors & Memory
Display
Removable Devices
Network Adapter
Hard Disk (SCSI)
CD/DVD (IDE)
Sound Card
USB & Bluetooth
Camera
Other
Startup Disk
Encryption
Compatibility
Isolation
Advanced

Choose FreeBSD ISO image or from a CD/DVD:

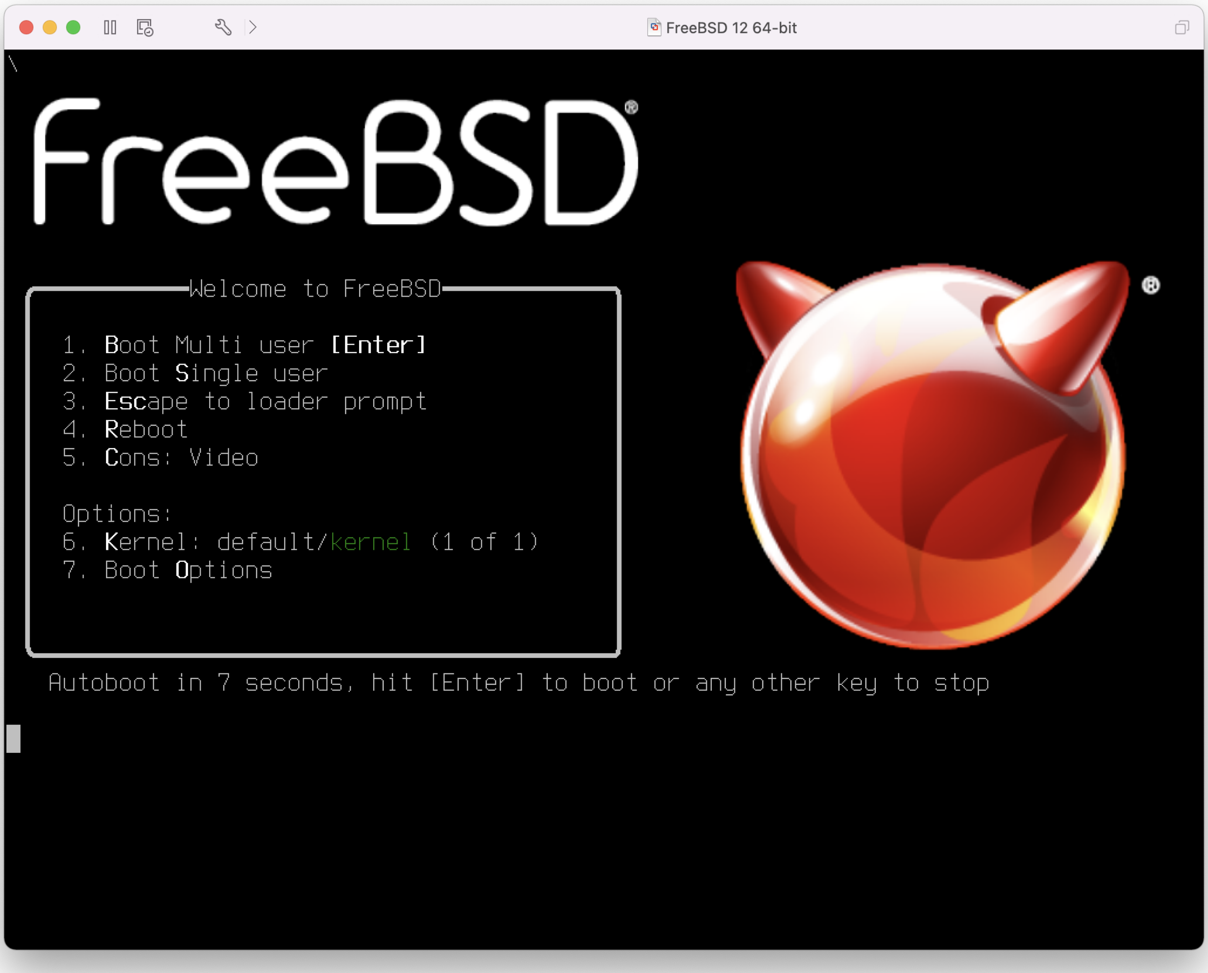

Show All
FreeBSD 12 64-bit: CD/DVD (IDE)
Add Device...
Advanced options
?

☑ Connect CD/DVD Drive

This CD/DVD drive is configured to use the following:

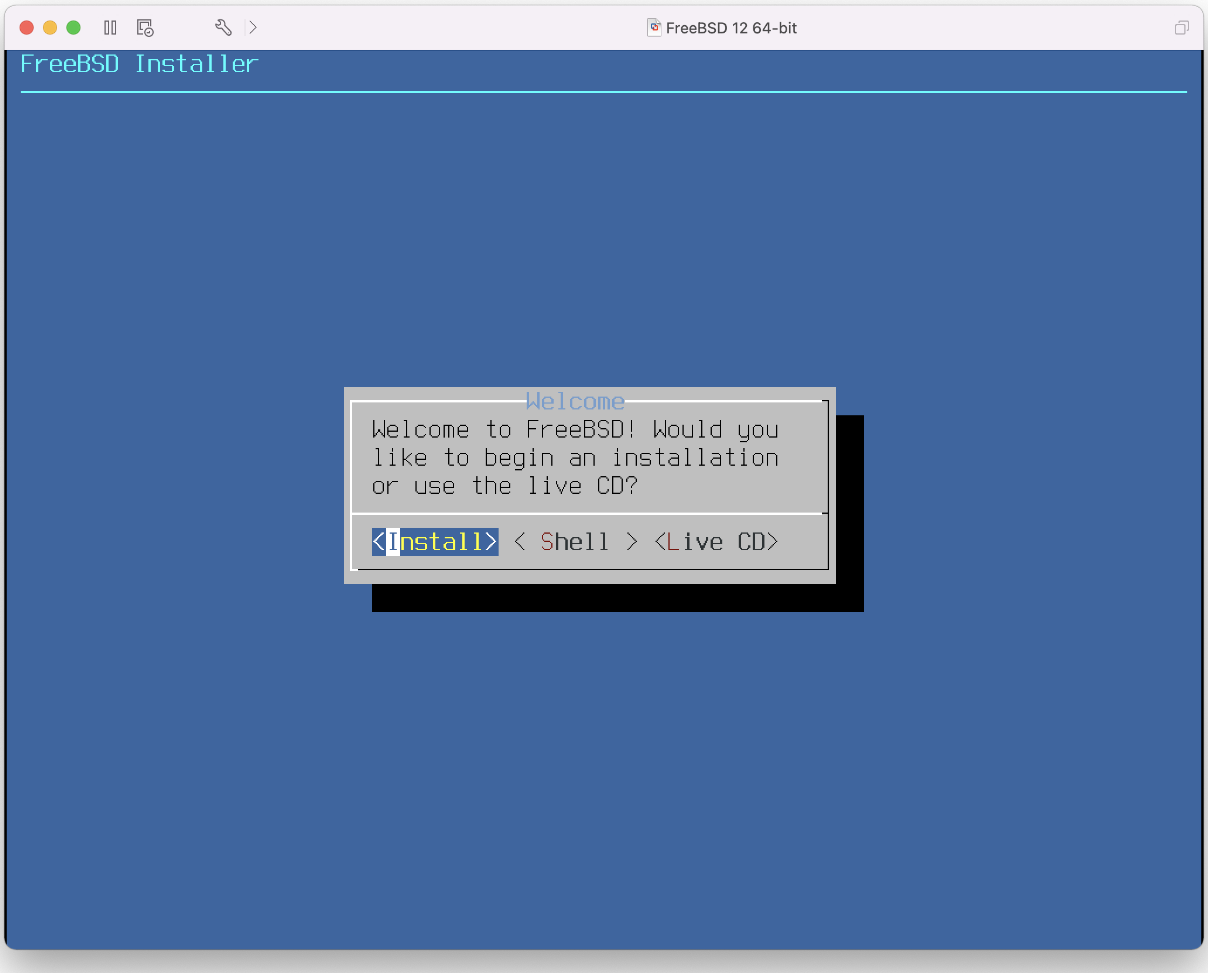
FreeBSD-13.0-RELEASE-amd64-dvd1.iso

Start the virtual machine:



```
Welcome to FreeBSD

 1. Boot Multi user [Enter]
 2. Boot Single user
 3. Escape to loader prompt
 4. Reboot
 5. Cons: Video

 Options:
 6. Kernel: default/kernel (1 of 1)
 7. Boot Options

Autoboot in 7 seconds, hit [Enter] to boot or any other key to stop
```

Install FreeBSD as usual:



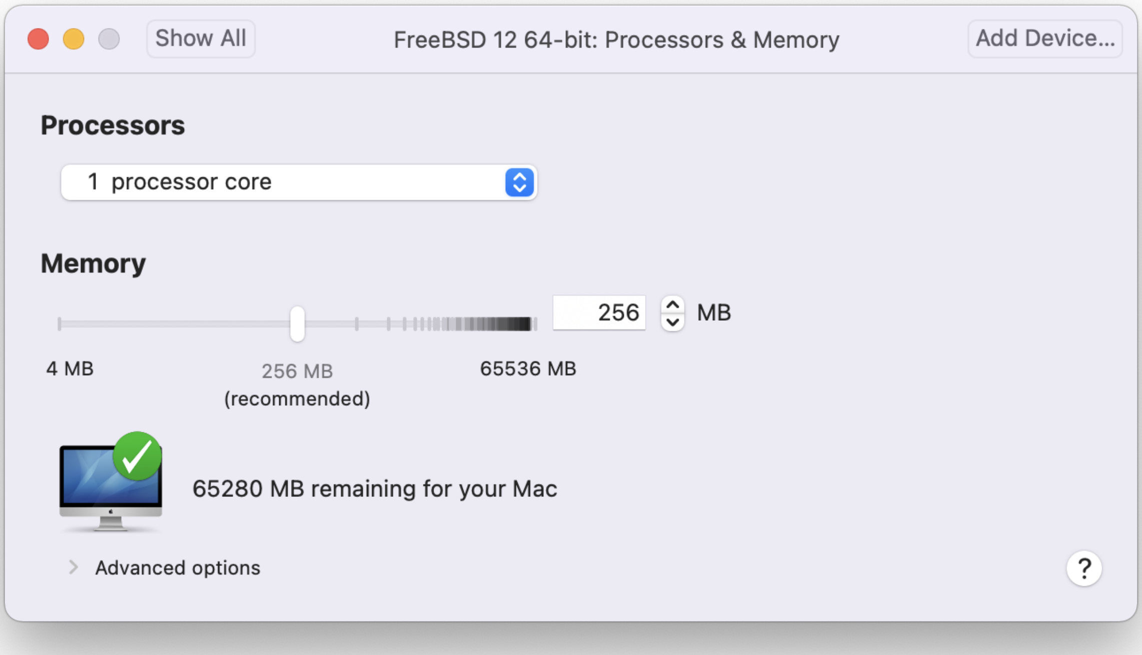
FreeBSD Installer

Welcome

Welcome to FreeBSD! Would you like to begin an installation or use the live CD?

Once the install is complete, the settings of the virtual machine can be modified, such as memory usage and the number of CPUs the virtual machine will have access to:

> ℹ The System Hardware settings of the virtual machine cannot be modified while the virtual machine is running.


Show All
Add Device...
4 MB
256 MB
(recommended)
65536 MB
Advanced options
?

FreeBSD 12 64-bit: Processors & Memory

Processors

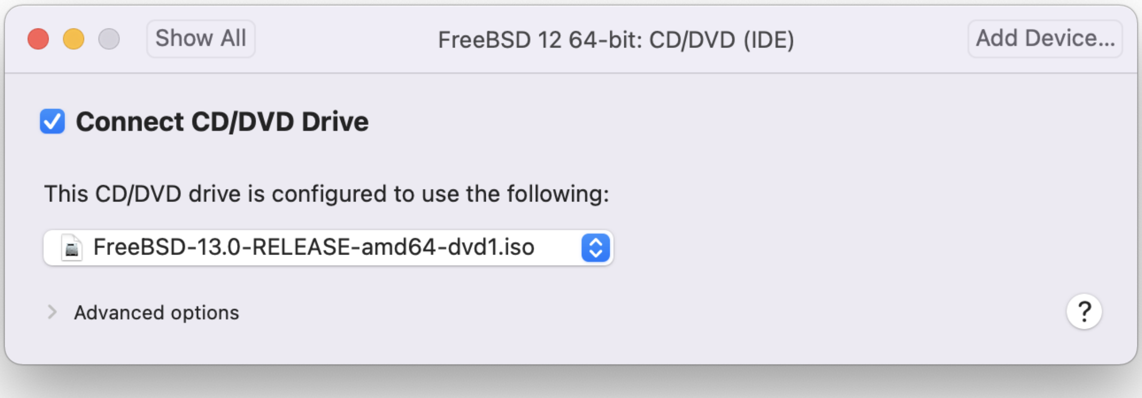
1 processor core

Memory

256 MB

65280 MB remaining for your Mac

The status of the CD-ROM device. Normally the CD/DVD/ISO is disconnected from the virtual machine when it is no longer needed.


Show All
Add Device...
Advanced options
?

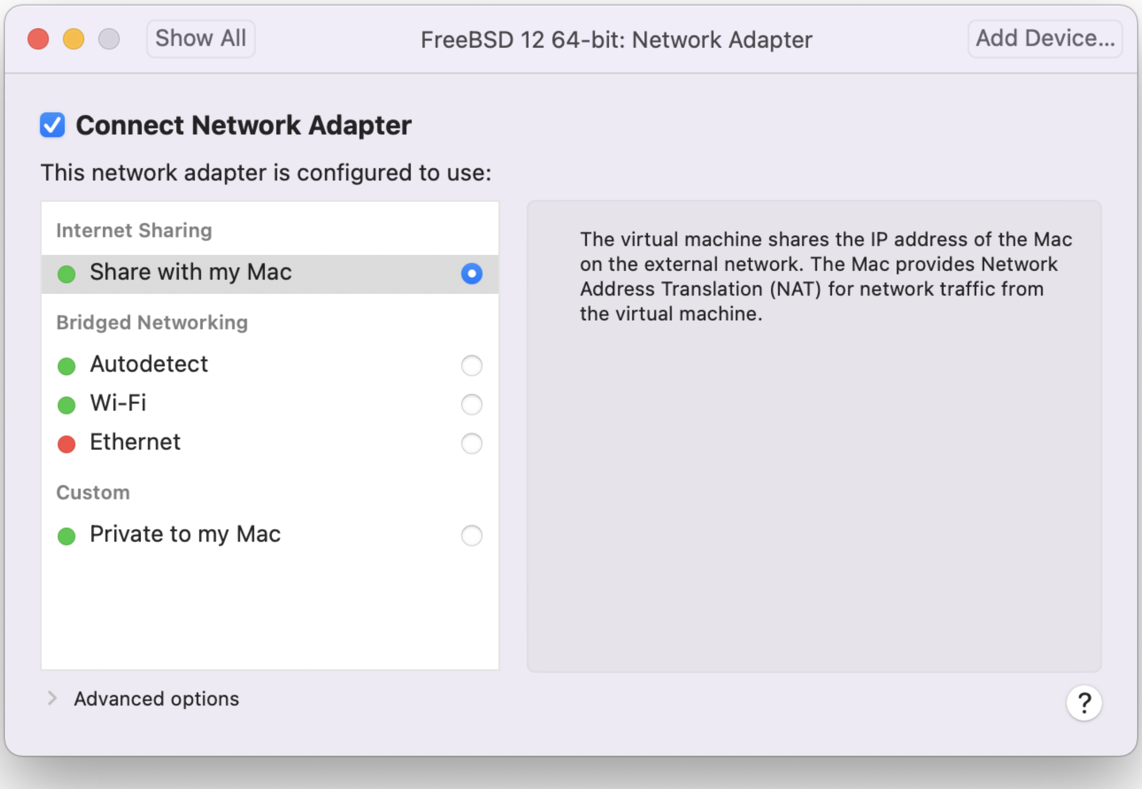
FreeBSD 12 64-bit: CD/DVD (IDE)

☑ **Connect CD/DVD Drive**

This CD/DVD drive is configured to use the following:

FreeBSD-13.0-RELEASE-amd64-dvd1.iso

The last thing to change is how the virtual machine will connect to the network. To allow connections to the virtual machine from other machines besides the host, choose Connect directly to the physical network (Bridged). Otherwise, Share the host’s internet connection (NAT) is preferred so that the virtual machine can have access to the Internet, but the network cannot access the virtual machine.



☑ **Connect Network Adapter**

This network adapter is configured to use:

Internet Sharing
- Share with my Mac ●

Bridged Networking
- Autodetect ○
- Wi-Fi ○
- Ethernet ○

Custom
- Private to my Mac ○

The virtual machine shares the IP address of the Mac on the external network. The Mac provides Network Address Translation (NAT) for network traffic from the virtual machine.

After modifying the settings, boot the newly installed FreeBSD virtual machine.

25.3.2. Configuring FreeBSD on VMware Fusion

After FreeBSD has been successfully installed on macOS® X with VMware Fusion, there are a number of configuration steps that can be taken to optimize the system for virtualized operation.

1. Set Boot Loader Variables

   The most important step is to reduce the `kern.hz` tunable to reduce the CPU utilization of FreeBSD under the VMware Fusion environment. This is accomplished by adding the following line to /boot/loader.conf:

   ```
   kern.hz=100
   ```

   Without this setting, an idle FreeBSD VMware Fusion guest will use roughly 15% of the CPU of a single processor iMac®. After this change, the usage will be closer to 5%.

2. Create a New Kernel Configuration File

   All FireWire, and USB device drivers can be removed from a custom kernel configuration file. VMware Fusion provides a virtual network adapter used by the em(4) driver, so all network devices except for em(4) can be removed from the kernel.

3. Configure Networking

The most basic networking setup uses DHCP to connect the virtual machine to the same local area network as the host Mac®. This can be accomplished by adding `ifconfig_em0="DHCP"` to /etc/rc.conf. More advanced networking setups are described in Advanced Networking.

4. Install drivers and open-vm-tools

   To run FreeBSD smoothly on VMWare, drivers should be installed:

   ```
   # pkg install xf86-video-vmware xf86-input-vmmouse open-vm-tools
   ```

   The xf86 packages are only available for x86 FreeBSD guests.

Enabling the mouse

+ Some users have reported issues using the mouse in the virtual machine. The mouse may be enabled by appending the following to /boot/loader.conf:

+

```
# ums_load="YES"
```

25.4. FreeBSD as a Guest on VirtualBox™

FreeBSD works well as a guest in VirtualBox™. The virtualization software is available for most common operating systems, including FreeBSD itself.

The VirtualBox™ guest additions provide support for:

- Clipboard sharing.
- Mouse pointer integration.
- Host time synchronization.
- Window scaling.
- Seamless mode.

For running Xorg, make sure to set **VBoxSVGA** for the Screen setting of the Display section in the virtual machine configuration before installing the Guest Additions. Do not select VMSVGA as that targets VMware guests and is not usable on FreeBSD. It causes Xorg to not run correctly. VBoxSVGA does not have this problem and executes X11 without errors.

These commands are run in the FreeBSD guest.

Install the emulators/virtualbox-ose-additions package or port in the FreeBSD guest. This will install the port:

```
# cd /usr/ports/emulators/virtualbox-ose-additions && make install clean
```

Add these lines to /etc/rc.conf:

```
vboxguest_enable="YES"
vboxservice_enable="YES"
```

If ntpd(8) or ntpdate(8) is used, disable host time synchronization:

```
vboxservice_flags="--disable-timesync"
```

Xorg will automatically recognize the `vboxvideo` driver. It can also be manually entered in /etc/X11/xorg.conf:

```
Section "Device"
    Identifier "Card0"
    Driver "vboxvideo"
    VendorName "InnoTek Systemberatung GmbH"
    BoardName "VirtualBox Graphics Adapter"
EndSection
```

To use the `vboxmouse` driver, adjust the mouse section in /etc/X11/xorg.conf:

```
Section "InputDevice"
    Identifier "Mouse0"
    Driver "vboxmouse"
EndSection
```

Shared folders for file transfers between host and VM are accessible by mounting them using `mount_vboxvfs`. A shared folder can be created on the host using the VirtualBox GUI or via `vboxmanage`. For example, to create a shared folder called *myshare* under /mnt/bsdboxshare for the VM named *BSDBox*, run:

```
# vboxmanage sharedfolder add 'BSDBox' --name myshare --hostpath /mnt/bsdboxshare
```

Note that the shared folder name must not contain spaces. Mount the shared folder from within the guest system like this:

```
# mount_vboxvfs -w myshare /mnt
```

25.5. FreeBSD as a Host with VirtualBox™

VirtualBox™ is an actively developed, complete virtualization package, that is available for most operating systems including Windows®, macOS®, Linux® and FreeBSD. It is equally capable of running Windows® or UNIX®-like guests. It is released as open source software, but with closed-source components available in a separate extension pack. These components include support for USB 2.0 devices. More information may be found on the Downloads page of the VirtualBox™ wiki. Currently, these extensions are not available for FreeBSD.

25.5.1. Installing VirtualBox™

VirtualBox™ is available as a FreeBSD package or port in emulators/virtualbox-ose. The port can be installed using these commands:

```
# cd /usr/ports/emulators/virtualbox-ose
# make install clean
```

One useful option in the port's configuration menu is the `GuestAdditions` suite of programs. These provide a number of useful features in guest operating systems, like mouse pointer integration (allowing the mouse to be shared between host and guest without the need to press a special keyboard shortcut to switch) and faster video rendering, especially in Windows® guests. The guest additions are available in the **Devices** menu, after the installation of the guest is finished.

A few configuration changes are needed before VirtualBox™ is started for the first time. The port installs a kernel module in /boot/modules which must be loaded into the running kernel:

```
# kldload vboxdrv
```

To ensure the module is always loaded after a reboot, add this line to /boot/loader.conf:

```
vboxdrv_load="YES"
```

To use the kernel modules that allow bridged or host-only networking, add this line to /etc/rc.conf and reboot the computer:

```
vboxnet_enable="YES"
```

The `vboxusers` group is created during installation of VirtualBox™. All users that need access to VirtualBox™ will have to be added as members of this group. `pw` can be used to add new members:

```
# pw groupmod vboxusers -m username
```

The default permissions for /dev/vboxnetctl are restrictive and need to be changed for bridged networking:

```
# chown root:vboxusers /dev/vboxnetctl
# chmod 0660 /dev/vboxnetctl
```

To make this permissions change permanent, add these lines to /etc/devfs.conf:

```
own     vboxnetctl root:vboxusers
perm    vboxnetctl 0660
```

To launch VirtualBox™, type from an Xorg session:

```
% VirtualBox
```

For more information on configuring and using VirtualBox™, refer to the official website. For FreeBSD-specific information and troubleshooting instructions, refer to the relevant page in the FreeBSD wiki.

25.5.2. VirtualBox™ USB Support

VirtualBox™ can be configured to pass USB devices through to the guest operating system. The host controller of the OSE version is limited to emulating USB 1.1 devices until the extension pack supporting USB 2.0 and 3.0 devices becomes available on FreeBSD.

For VirtualBox™ to be aware of USB devices attached to the machine, the user needs to be a member of the `operator` group.

```
# pw groupmod operator -m username
```

Then, add the following to /etc/devfs.rules, or create this file if it does not exist yet:

```
[system=10]
add path 'usb/*' mode 0660 group operator
```

To load these new rules, add the following to /etc/rc.conf:

```
devfs_system_ruleset="system"
```

Then, restart devfs:

```
# service devfs restart
```

Restart the login session and VirtualBox™ for these changes to take effect, and create USB filters as necessary.

25.5.3. VirtualBox™ Host DVD/CD Access

Access to the host DVD/CD drives from guests is achieved through the sharing of the physical drives. Within VirtualBox™, this is set up from the Storage window in the Settings of the virtual machine. If needed, create an empty IDECD/DVD device first. Then choose the Host Drive from the popup menu for the virtual CD/DVD drive selection. A checkbox labeled `Passthrough` will appear. This allows the virtual machine to use the hardware directly. For example, audio CDs or the burner will only function if this option is selected.

In order for users to be able to use VirtualBox™DVD/CD functions, they need access to /dev/xpt0, /dev/cdN, and /dev/passN. This is usually achieved by making the user a member of `operator`. Permissions to these devices have to be corrected by adding these lines to /etc/devfs.conf:

```
perm cd* 0660
perm xpt0 0660
perm pass* 0660
```

```
# service devfs restart
```

25.6. Virtualization with QEMU on FreeBSD

QEMU is a generic machine emulator and virtualizer that is completely open source software. It is developed by a large, active community and provides support for FreeBSD, OpenBSD, and NetBSD as well as other operating systems.

From the QEMU documentation:

- QEMU can be used in several different ways. The most common is for System Emulation, where it provides a virtual model of an entire machine (CPU, memory, and emulated devices) to run a guest OS. In this mode the CPU may be fully emulated, or it may work with a hypervisor such as `KVM`, `Xen` or `Hypervisor.Framework` to allow the guest to run directly on the host CPU.
- The second supported way to use QEMU is User Mode Emulation, where QEMU can launch processes compiled for one CPU on another CPU. In this mode the CPU is always emulated.
- QEMU also provides a number of standalone command line utilities, such as the qemu-img(1) disk image utility that allows one to create, convert, and modify disk images.

QEMU can emulate a wide number of architectures including `Arm™`, `i386`, `x86_64`, `MIPS™`, `s390X`, `SPARC™` (Sparc™ and Sparc64™), and others. The list of QEMU System Emulator Targets is regularly kept up to date.

This section describes how to use QEMU for both System Emulation and User Mode Emulation on FreeBSD, and provides examples of using QMEU commands and command line utilities.

25.6.1. Installing QEMU Software

QEMU is available as a FreeBSD package or as a port in emulators/qemu. The package build includes sane options and defaults for most users and is the recommended method of installation.

```
# pkg install qemu
```

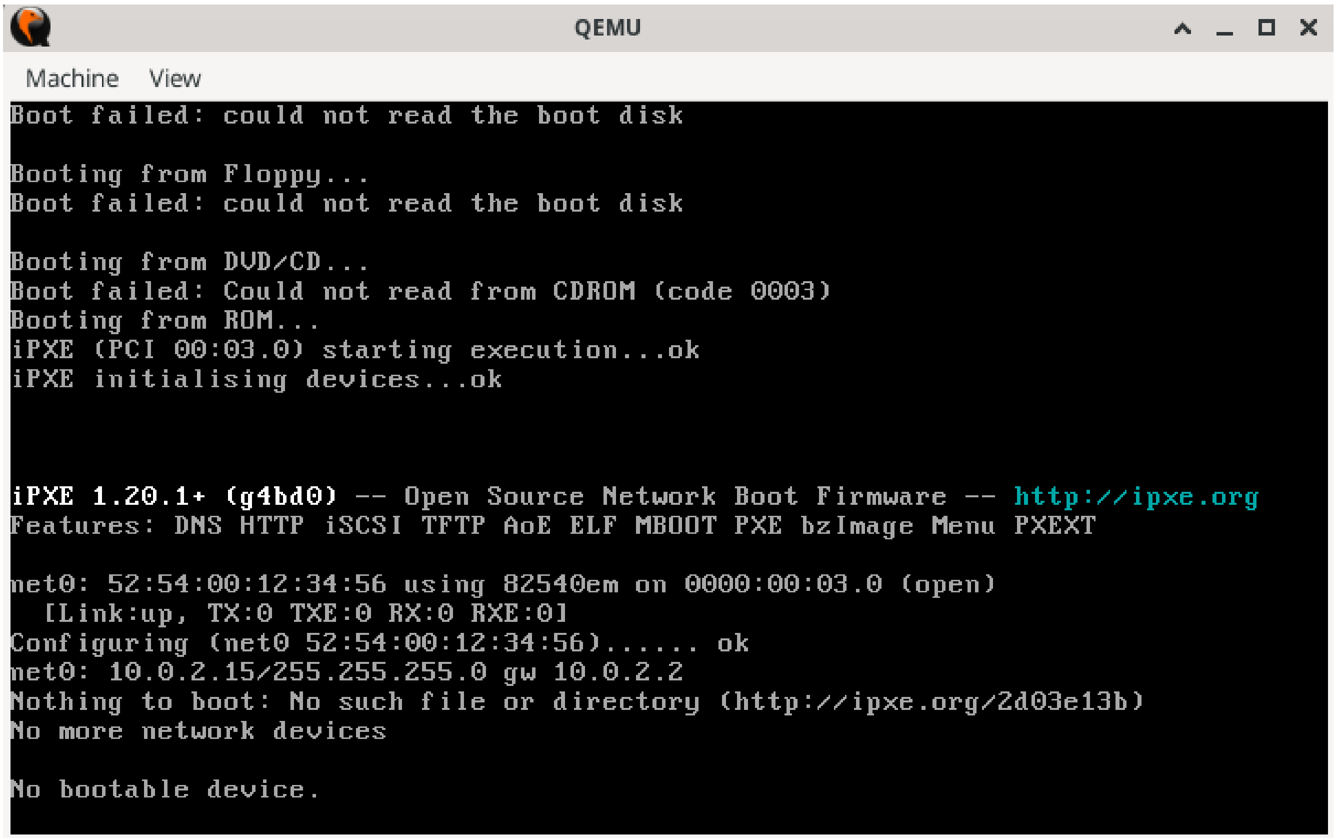
The package installation includes several dependencies. Once the installation is complete, create a link to the host version of QEMU that will be used most often. If the host is an Intel™ or AMD™ 64 bit system that will be:

```
# ln -s /usr/local/bin/qemu-system-x86_64 /usr/local/bin/qemu
```

Test the installation by running the following command as a non-root user:

```
% qemu
```

This brings up a window with QEMU actively trying to boot from hard disk, floppy disk, DVD/CD, and PXE. Nothing has been set up yet, so the command will produce several errors and end with "No bootable device" as shown in Figure 1. However, it does show that the QEMU software has been installed correctly.



```
Boot failed: could not read the boot disk

Booting from Floppy...
Boot failed: could not read the boot disk

Booting from DVD/CD...
Boot failed: Could not read from CDROM (code 0003)
Booting from ROM...
iPXE (PCI 00:03.0) starting execution...ok
iPXE initialising devices...ok



iPXE 1.20.1+ (g4bd0) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 52:54:00:12:34:56 using 82540em on 0000:00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 52:54:00:12:34:56)...... ok
net0: 10.0.2.15/255.255.255.0 gw 10.0.2.2
Nothing to boot: No such file or directory (http://ipxe.org/2d03e13b)
No more network devices

No bootable device.
```

Figure 61. QEMU with no bootable image

25.6.2. Virtual Machine Install

QEMU is under very active development. Features and command options can change from one version to the next. This section provides examples developed

> with QEMU version 9.0.1 (Summer, 2024). When in doubt, always consult the QEMU Documentation particularly the About QEMU page which has links to supported build platforms, emulation, deprecated features, and removed features.

Follow the steps below to create two virtual machines named "**left**", and "**right**". Most commands can be performed without root privileges.

1. Create a test environment to work with QEMU:

   ```
   % mkdir -p ~/QEMU  ~/QEMU/SCRIPTS  ~/QEMU/ISO  ~/QEMU/VM
   ```

   The SCRIPTS directory is for startup scripts and utilities. The ISO directory is for the guest ISO boot images. The VM directory is where the virtual machine images (`VMs`) will reside.

2. Download a recent copy of FreeBSD into ~/QEMU/ISO:

   ```
   % cd ~/QEMU/ISO
   % fetch https://download.freebsd.org/releases/ISO-IMAGES/14.1/FreeBSD-14.1-RELEASE-
   amd64-bootonly.iso
   ```

   Once the download is complete create a shorthand link. This shorthand link is used in the startup scripts below.

   ```
   % ln -s FreeBSD-14.1-RELEASE-amd64-bootonly.iso  fbsd.iso
   ```

3. Change directory to the location for virtual machines (~/QEMU/VM). Run qemu-img(1) to create the disk images for the “left” VM:

   ```
   % cd ~/QEMU/VM
   % qemu-img create -f raw  left.img   15G
   ```

   The QEMU `raw` format is designed for performance. The format is straightforward and has no overhead which makes it faster, especially for high performance or high throughput scenarios. The use case is for maximum performance where no additional features, such as snapshots, are needed. This format is used in the script for the "left" VM below.

   A separate format is `qcow2` which uses QEMU’s "copy on write" technique for managing disk space. This technique does not require a complete 15G disk, just a stub version that is managed directly by the VM. The disk grows dynamically as the VM writes to it. This format supports snapshots, compression, and encryption. The use case for this format is for development, testing, and scenarios with the need of these advanced features. This format is used in the script for the "right" VM below.

   Run qemu-img(1) again to create the disk image for the "right" VM using `qcow2`:

```
% qemu-img create -f qcow2 -o preallocation=full,cluster_size=512K,lazy_refcounts
=on right.qcow2 20G
```

To see the actual size of the file use:

```
% du -Ah right.qcow2
```

4. Set up networking for both virtual machines with the following commands. In this example the host network interface is `em0`. If necessary, change it to fit the interface for the host system. This must be done after every host machine restart to enable the QEMU guest VMs to communicate.

```
# ifconfig tap0 create
# ifconfig tap1 create
# sysctl net.link.tap.up_on_open=1
net.link.tap.up_on_open: 0 -> 1
# sysctl net.link.tap.user_open=1
net.link.tap.user_open: 0 -> 1
# ifconfig bridge0 create
# ifconfig bridge0 addm tap0 addm tap1 addm em0
# ifconfig bridge0 up
```

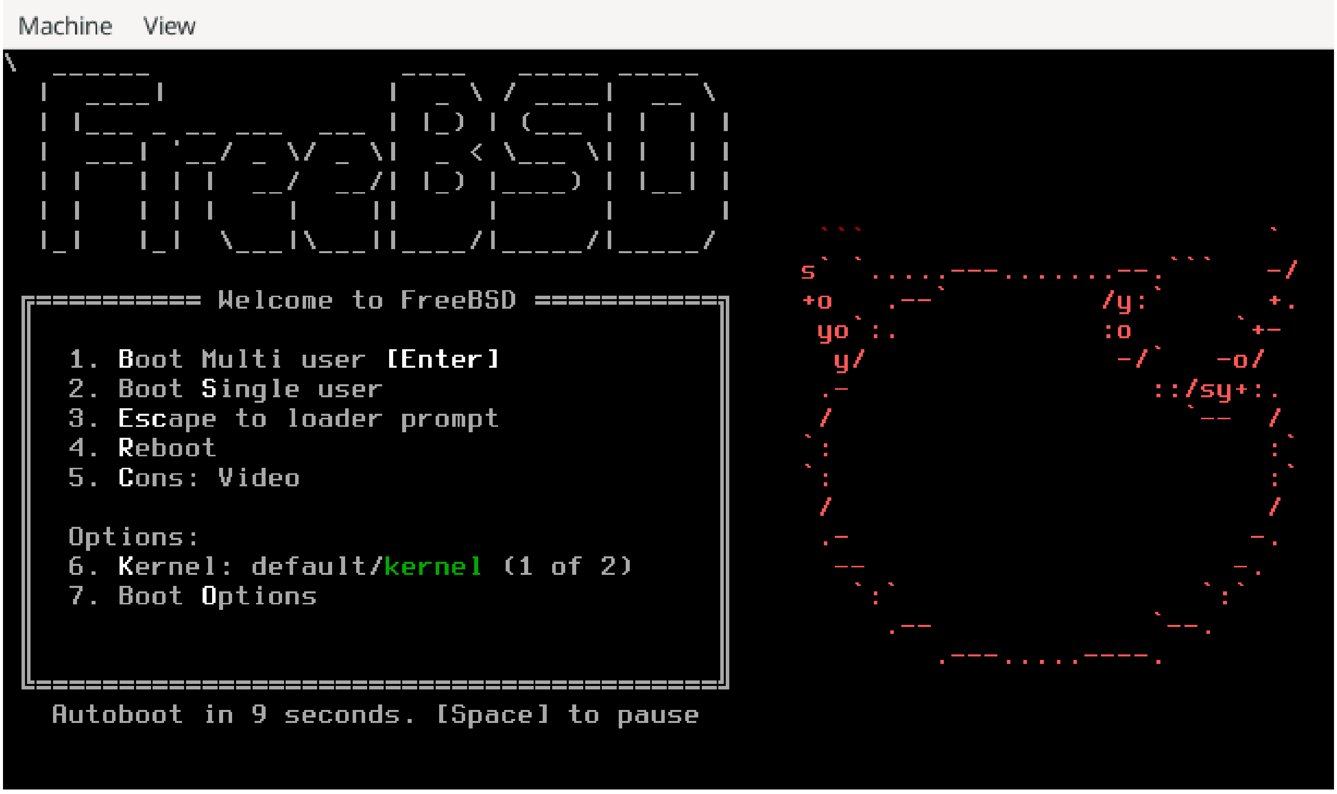
The above commands create two tap(4) devices (`tap0`, `tap1`) and one if_bridge(4) device (`bridge0`). Then, they add the `tap` devices and the local host interface (`em0`) to the `bridge`, and set two sysctl(8) entries to allow for normal users to open the tap device. These commands will allow the virtual machines to talk to the network stack on the host.

5. Change to ~/QEMU/SCRIPTS, use the following script to start the first virtual machine, "left". This script uses the QEMU raw disk format.

```
/usr/local/bin/qemu-system-x86_64  -monitor none \
  -cpu qemu64 \
  -vga std \
  -m 4096 \
  -smp 4   \
  -cdrom ../ISO/fbsd.iso \
  -boot order=cd,menu=on \
  -blockdev driver=file,aio=threads,node-name=imgleft,filename=../VM/left.img \
  -blockdev driver=raw,node-name=drive0,file=imgleft \
  -device virtio-blk-pci,drive=drive0,bootindex=1  \
  -netdev tap,id=nd0,ifname=tap0,script=no,downscript=no,br=bridge0 \
  -device e1000,netdev=nd0,mac=02:20:6c:65:66:74 \
  -name \"left\"
```

Save the above into a file (for example `left.sh`) and simply run: % `/bin/sh left.sh`

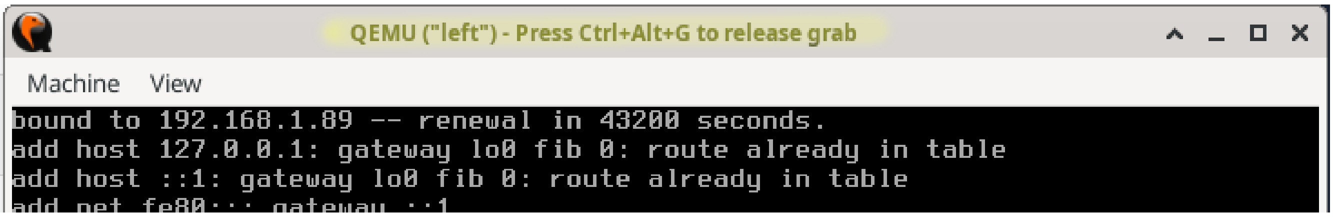
QEMU will start up a virtual machine in a separate window and boot the FreeBSD iso as shown in Figure 2. All command options such as `-cpu` and `-boot` are fully described in the QEMU man page qemu(1).



```
Welcome to FreeBSD

 1. Boot Multi user [Enter]
 2. Boot Single user
 3. Escape to loader prompt
 4. Reboot
 5. Cons: Video

 Options:
 6. Kernel: default/kernel (1 of 2)
 7. Boot Options

Autoboot in 9 seconds. [Space] to pause
```

Figure 62. FreeBSD Boot Loader Menu

> If the mouse is clicked in the QEMU console window, QEMU will “grab” the mouse as shown in Figure 4. Type `Ctl`+`Alt`+`G`” to release the mouse.



```
bound to 192.168.1.89 -- renewal in 43200 seconds.
add host 127.0.0.1: gateway lo0 fib 0: route already in table
add host ::1: gateway lo0 fib 0: route already in table
add net fe80::: gateway ::1
```

Figure 63. When QEMU Has Grabbed the Mouse

> On FreeBSD, an initial QEMU installation can be somewhat slow. This is because the emulator writes filesystem formatting and metadata during the disk first use. Subsequent operations are generally much faster.

During the installation there are several points to note:

- Select to use UFS as the filesystem. ZFS does not perform well with small memory sizes.
- For networking use DHCP. If desired, configure IPv6 if supported by the local LAN.
- When adding the default user, ensure they are a member of the **wheel** group.

Once the installation completes, the virtual machine reboots into the newly installed FreeBSD image.

Login as `root` and update the system as follows:

```
# freebsd-update fetch
# freebsd-update install
# shutdown -r now
```

After a successful installation, QEMU will boot the operating system installed on the disk, and not the installation program.

QEMU supports a `-runas` option. For added security, include the option "-runas user_name" in the script listing above. See qemu(1) for details.

Login as `root` again and add any packages desired. To utilize the X Window system in the guest, see the section "Using the X Window System" below.

This completes the setup of the "left" VM.

To install the "right" VM, run the following script. This script has the modifications needed for tap1, format=qcow2, the image filename, the MAC address, and the terminal window name. If desired, include the "-runas" parameter as described in the above note.

```
/usr/local/bin/qemu-system-x86_64  -monitor none \
  -cpu qemu64 \
  -vga cirrus \
  -m 4096  -smp 4   \
  -cdrom ../ISO/fbsd.iso \
  -boot order=cd,menu=on \
  -drive
if=none,id=drive0,cache=writeback,aio=threads,format=qcow2,discard=unmap,file=../VM/ri
ght.qcow2 \
  -device virtio-blk-pci,drive=drive0,bootindex=1  \
  -netdev tap,id=nd0,ifname=tap1,script=no,downscript=no,br=bridge0 \
  -device e1000,netdev=nd0,mac=02:72:69:67:68:74 \
  -name \"right\"
```

Once the installation is complete, the "left" and "right" machines can communicate with each other and with the host. If there are strict firewall rules on the host, consider adding or modifying rules to allow the bridge and tap devices to communicate with each other.

25.6.3. Usage Tips

Using the X Window System

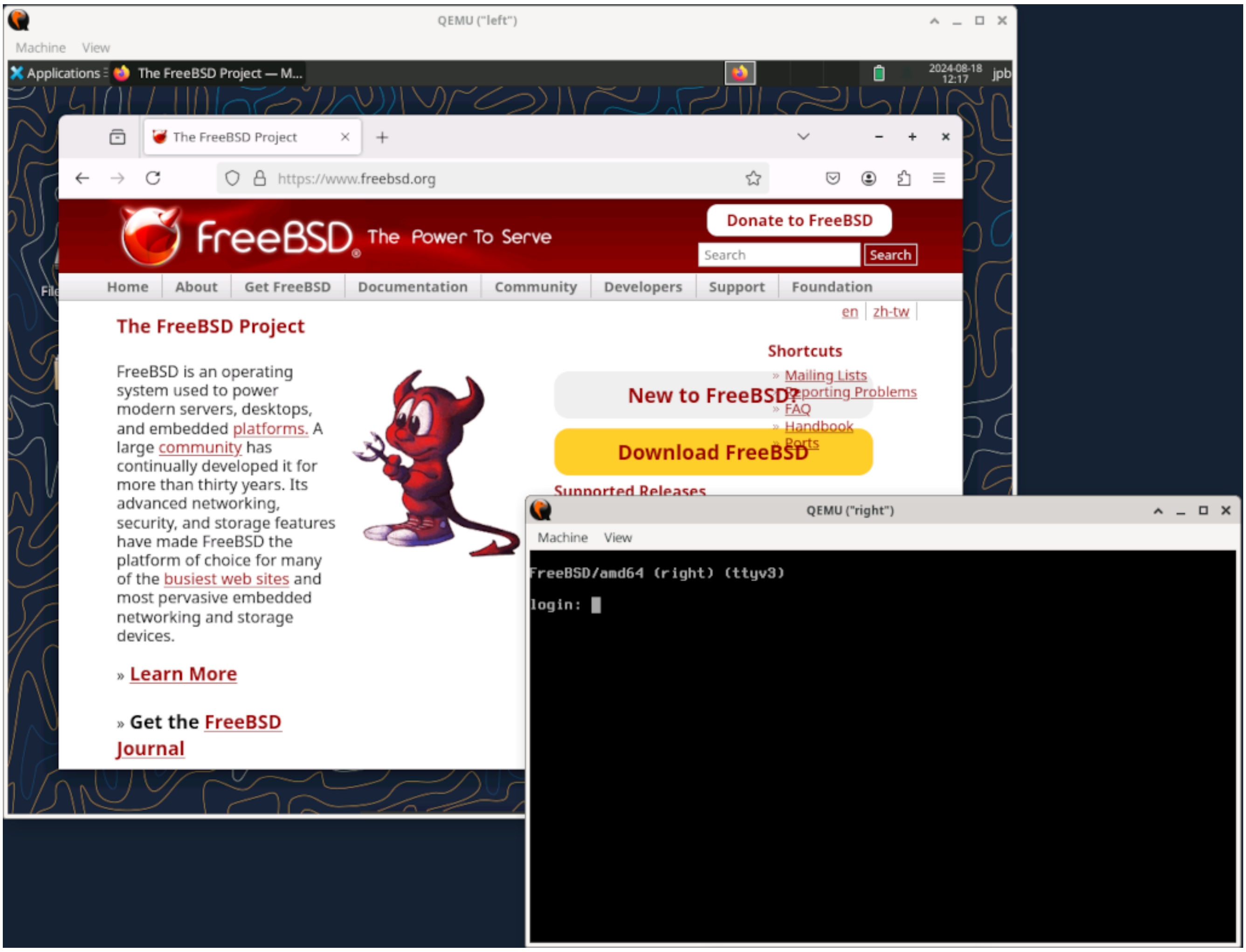
Installing Xorg describes how to set up the `X Window` system. Refer to that guide for initial `X Window` setup then consult Desktop Environments on how to set up a complete desktop.

This section demonstrates use of the XFCE desktop.

Once the installation is complete, login as a regular user, then type:

```
% startx
```

The XFCE4 window manager should start up and present a functioning graphical desktop as in Figure 5. On initial startup, it may take up to a minute to display the desktop. See the documentation at the XFCE website for usage details.


QEMU ("left")
Machine
View
Applications
The FreeBSD Project — M...
2024-08-18
12:17
jpb
The FreeBSD Project
https://www.freebsd.org
FreeBSD
The Power To Serve
Donate to FreeBSD
Search
Search
Home
About
Get FreeBSD
Documentation
Community
Developers
Support
Foundation
en
zh-tw
Shortcuts
Mailing Lists
Reporting Problems
FAQ
Handbook
Ports
New to FreeBSD?
Download FreeBSD
Supported Releases
» Learn More
» Get the FreeBSD Journal
QEMU ("right")
Machine
View

The FreeBSD Project

FreeBSD is an operating system used to power modern servers, desktops, and embedded platforms. A large community has continually developed it for more than thirty years. Its advanced networking, security, and storage features have made FreeBSD the platform of choice for many of the busiest web sites and most pervasive embedded networking and storage devices.

```
FreeBSD/amd64 (right) (ttyv3)

login:
```

Figure 64. Both QEMU VMs

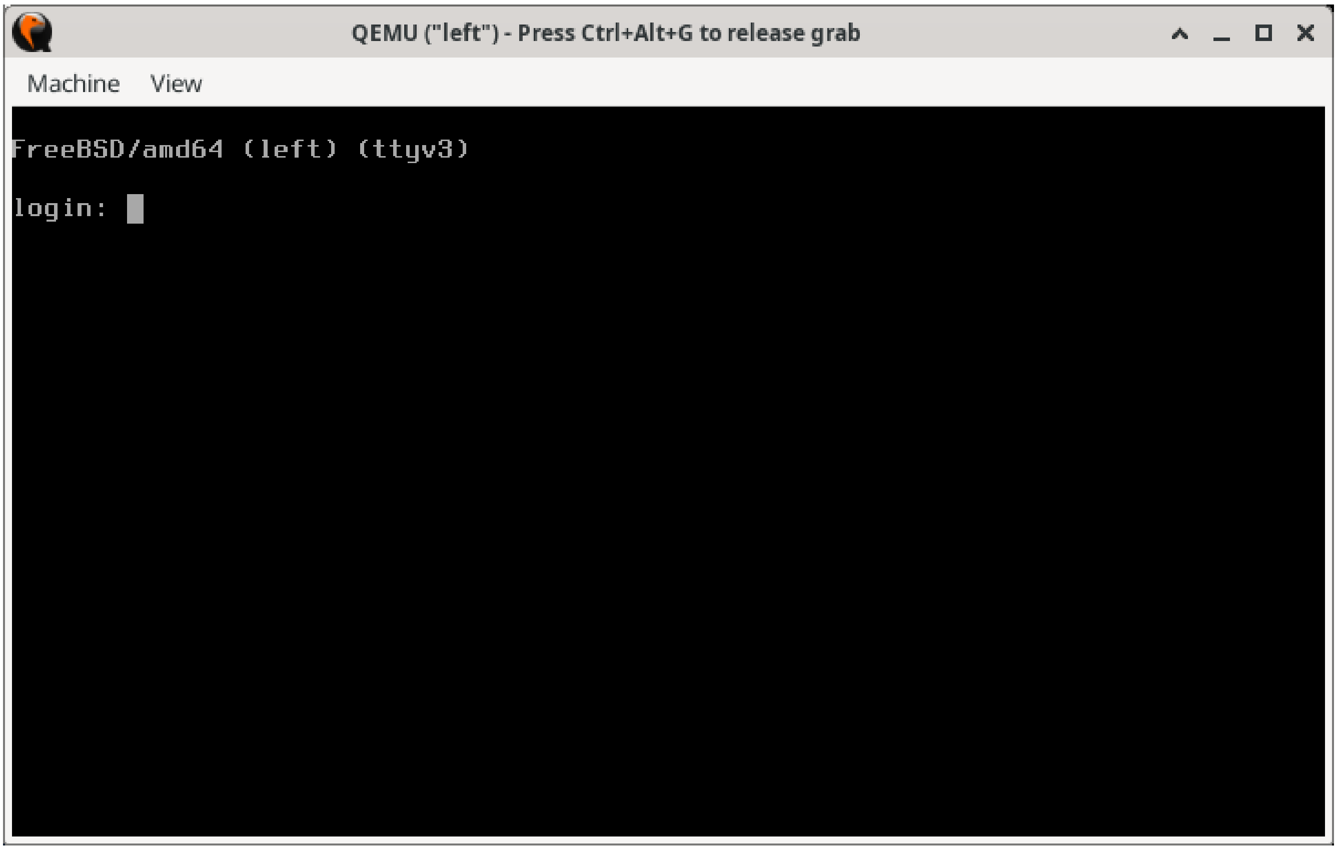
> Adding more memory to the guest system may speed up the graphical user interface.

Here, the "left" VM has had the `X Window` system installed, while the "right" VM is still in text mode.

Using the QEMU Window

The QEMU window functions as a full FreeBSD console, and is capable of running multiple virtual terminals, just like a bare-metal system.

To switch to another virtual console, click into the QEMU window and type `Alt`+`F2` or `Alt`+`F3`. FreeBSD should switch to another virtual console. Figure 6 shows the "left" VM displaying the virtual console on `ttyv3`.



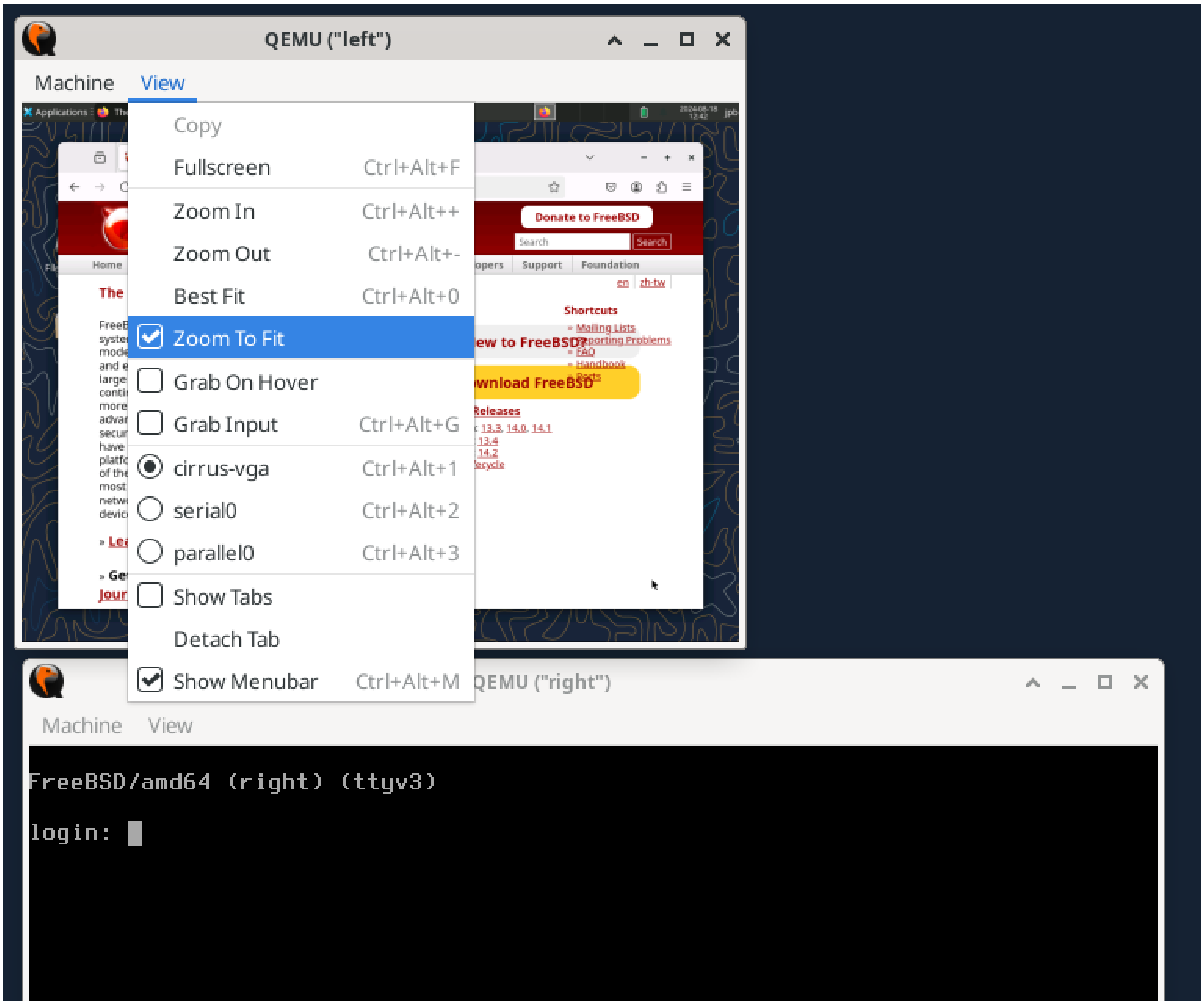
```
FreeBSD/amd64 (left) (ttyv3)

login:
```

Figure 65. Switching to Another Virtual Console in the QEMU Window

The host current desktop manager or window manager may be already setup for another function with the `Alt` + `F1`, `Alt` + `F2` key sequences. If so, try typing `Ctl` + `Alt` + `F1`, `Ctl` + `Alt` + `F2`, or some other similar key combination. Check the window manager or desktop manager documentation for details.

Using the QEMU Window Menus

Another feature of the QEMU window is the `View` menu and the Zoom controls. The most useful is `Zoom to Fit`. When this menu selection is clicked, it is then possible to resize the QEMU window by clicking the window corner controls and resizing the window. Figure 7 shows the effect of resizing the "left" window while in graphics mode.


QEMU ("left")
Machine
View
Copy
Fullscreen Ctrl+Alt+F
Zoom In Ctrl+Alt++
Zoom Out Ctrl+Alt+-
Best Fit Ctrl+Alt+0
☑ Zoom To Fit
☐ Grab On Hover
☐ Grab Input Ctrl+Alt+G
● cirrus-vga Ctrl+Alt+1
○ serial0 Ctrl+Alt+2
○ parallel0 Ctrl+Alt+3
☐ Show Tabs
Detach Tab
☑ Show Menubar Ctrl+Alt+M
QEMU ("right")
Machine
View

```
FreeBSD/amd64 (right) (ttyv3)

login:
```

Figure 66. Using the View Menu `Zoom to Fit` Option

Other QEMU Window Menu Options

Also shown in the `View` menu are

- `cirrus-vga`, `serial0`, and `parallel0` options. These allow for switching input/output to the selected device.

The QEMU window `Machine` menu allows for four types of control over the guest VM:

- `Pause` allows for pausing the QEMU virtual machine. This may be helpful in freezing a fast scrolling window.
- `Reset` immediately resets the virtual machine back at cold "power on" state. As with a real machine, it is not recommended unless absolutely necessary.
- `Power Down` simulates an ACPI shutdown signal and the operating system goes through a graceful shutdown.
- `Quit` powers off the virtual machine immediately - also not recommended unless necessary.

25.6.4. Adding a Serial Port Interface to a Guest VM

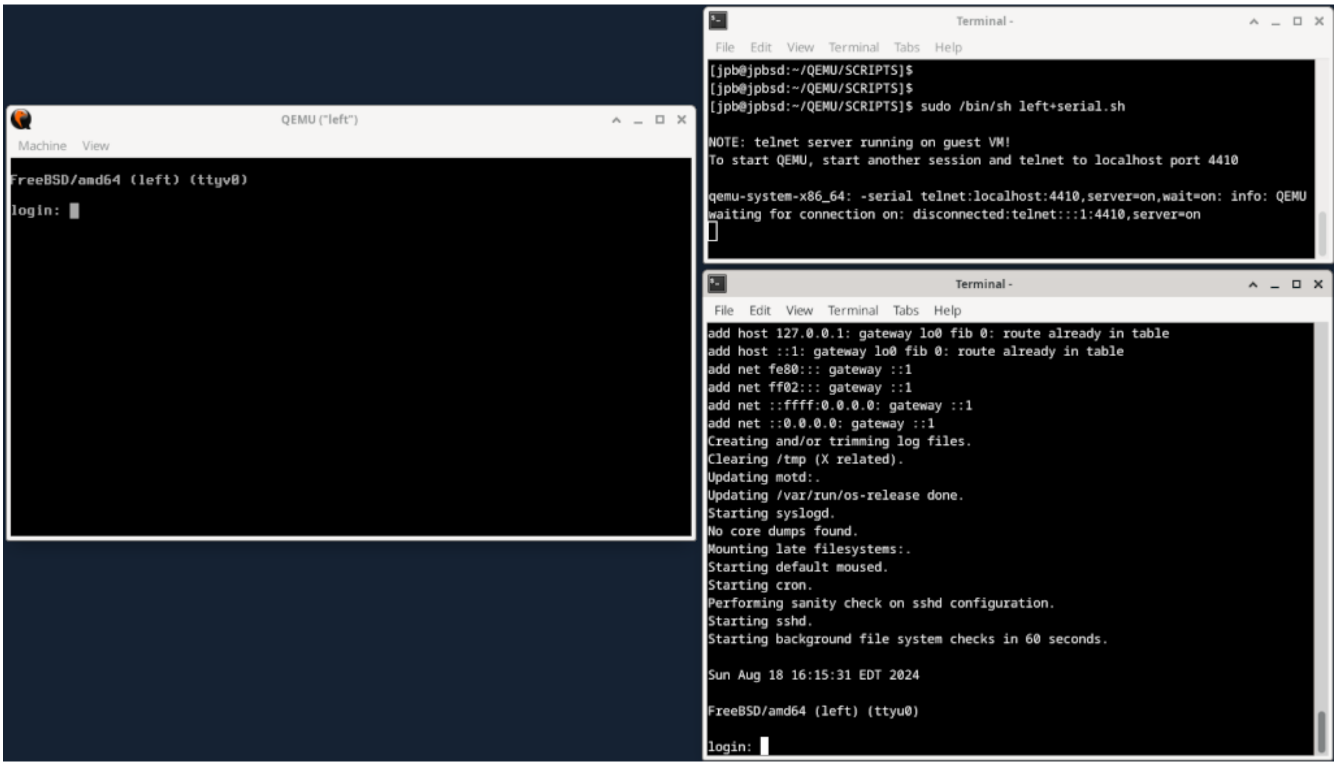
To implement a serial console, a guest VM running FreeBSD needs to insert

```
console="comconsole"
```

in /boot/loader.conf to allow the use of the FreeBSD serial console.

The updated configuration below shows how to implement the serial console on the guest VM. Run the script to start the VM.

```
# left+serial.sh
echo
echo "NOTE: telnet startup server running on guest VM!"
echo "To start QEMU, start another session and telnet to localhost port 4410"
echo

/usr/local/bin/qemu-system-x86_64  -monitor none \
  -serial telnet:localhost:4410,server=on,wait=on\
  -cpu qemu64 \
  -vga std \
  -m 4096 \
  -smp 4   \
  -cdrom ../ISO/fbsd.iso \
  -boot order=cd,menu=on \
  -blockdev driver=file,aio=threads,node-name=imgleft,filename=../VM/left.img \
  -blockdev driver=raw,node-name=drive0,file=imgleft \
  -device virtio-blk-pci,drive=drive0,bootindex=1  \
  -netdev tap,id=nd0,ifname=tap0,script=no,downscript=no,br=bridge0 \
  -device e1000,netdev=nd0,mac=02:20:6c:65:66:74 \
  -name \"left\"
```



```
FreeBSD/amd64 (left) (ttyv0)

login:
```

```
[jpb@jpbsd:~/QEMU/SCRIPTS]$
[jpb@jpbsd:~/QEMU/SCRIPTS]$
[jpb@jpbsd:~/QEMU/SCRIPTS]$ sudo /bin/sh left+serial.sh

NOTE: telnet server running on guest VM!
To start QEMU, start another session and telnet to localhost port 4410

qemu-system-x86_64: -serial telnet:localhost:4410,server=on,wait=on: info: QEMU
waiting for connection on: disconnected:telnet:::1:4410,server=on
```

```
add host 127.0.0.1: gateway lo0 fib 0: route already in table
add host ::1: gateway lo0 fib 0: route already in table
add net fe80::: gateway ::1
add net ff02::: gateway ::1
add net ::ffff:0.0.0.0: gateway ::1
add net ::0.0.0.0: gateway ::1
Creating and/or trimming log files.
Clearing /tmp (X related).
Updating motd:.
Updating /var/run/os-release done.
Starting syslogd.
No core dumps found.
Mounting late filesystems:.
Starting default moused.
Starting cron.
Performing sanity check on sshd configuration.
Starting sshd.
Starting background file system checks in 60 seconds.

Sun Aug 18 16:15:31 EDT 2024

FreeBSD/amd64 (left) (ttyu0)

login:
```

Figure 67. Enabling a Serial Port over TCP

In Figure 8, the serial port is redirected to a TCP port on the host system at VM startup and the QEMU monitor waits (`wait=on`) to activate the guest VM until a telnet(1) connection occurs on the indicated localhost port. After receiving a connection from a separate session, the FreeBSD system starts booting and looks for a console directive in /boot/loader.conf. With the directive "console=comconsole", FreeBSD starts up a console session on a serial port. The QEMU monitor detects this and directs the necessary character I/O on that serial port to the telnet session on the host. The system boots and once finished, login prompts are enabled on the serial port (`ttyu0`) and on the console (`ttyv0`).

It is important to note that the this serial redirect over TCP takes place outside the virtual machine. There is no interaction with any network on the virtual machine and therefore it is not subject to any firewall rules. Think of it like a dumb terminal sitting on an RS-232 or USB port on a real machine.

Notes on Using the Serial Console

On the serial console, if the window is resized, execute resizewin(1) to update the terminal size.

It may be desirable (even necessary) to stop syslog message from being sent to the console (both the QEMU console and the serial port). Consult syslog.conf(5) for details on redirecting console messages.

> Once the /boot.loader.conf has been updated to permit a serial console, the guest VM will attempt to boot from the serial port every time. Ensure that the serial port is enabled as shown in the listing above, or update the /boot/loader.conf file to not require a serial console.

25.6.5. QEMU User Mode Emulation

QEMU also supports running applications that are precompiled on an architecture different from the host CPU. For example, it is possible to run a Sparc64 architecture operating system on an x86_64 host. This is demonstrated in the next section.

Setting up a SPARC64 Guest VM on an x86_64 Host

Setting up a new VM with an architecture different from the host involves several steps:

- Getting the software that will run on the guest VM
- Creating a new disk image for the guest VM
- Setting up a new QEMU script with the new architecture
- Performing the install

In the following procedure a copy of OpenBSD 6.8 SPARC64 software is used for this QEMU User Mode Emulation exercise.

Not all versions of OpenBSD Sparc64 work on QEMU. OpenBSD version 6.8 is known to work and was selected as the example for this section.

1. Download OpenBSD 6.8 Sparc64 from an OpenBSD archive.

   On the OpenBSD download sites, only the most current versions are maintained. It is necessary to search an archive to obtain past releases.

   ```
   % cd ~/QEMU/ISO
   % fetch https://mirror.planetunix.net/pub/OpenBSD-archive/6.8/sparc64/install68.iso
   ```

2. Creating a new disk image for the Sparc64 VM is similar to the "right" VM above. This case uses the QEMU qcow2 format for the disk:

   ```
   % cd ~/QEMU/VM
   qemu-img create -f qcow2 -o preallocation=full,lazy_refcounts=on sparc64.qcow2 16G
   ```

3. Use the script below for the new Sparc64 architecture. As with the above example, run the script, then start a new session and `telnet` to localhost on the port indicated:

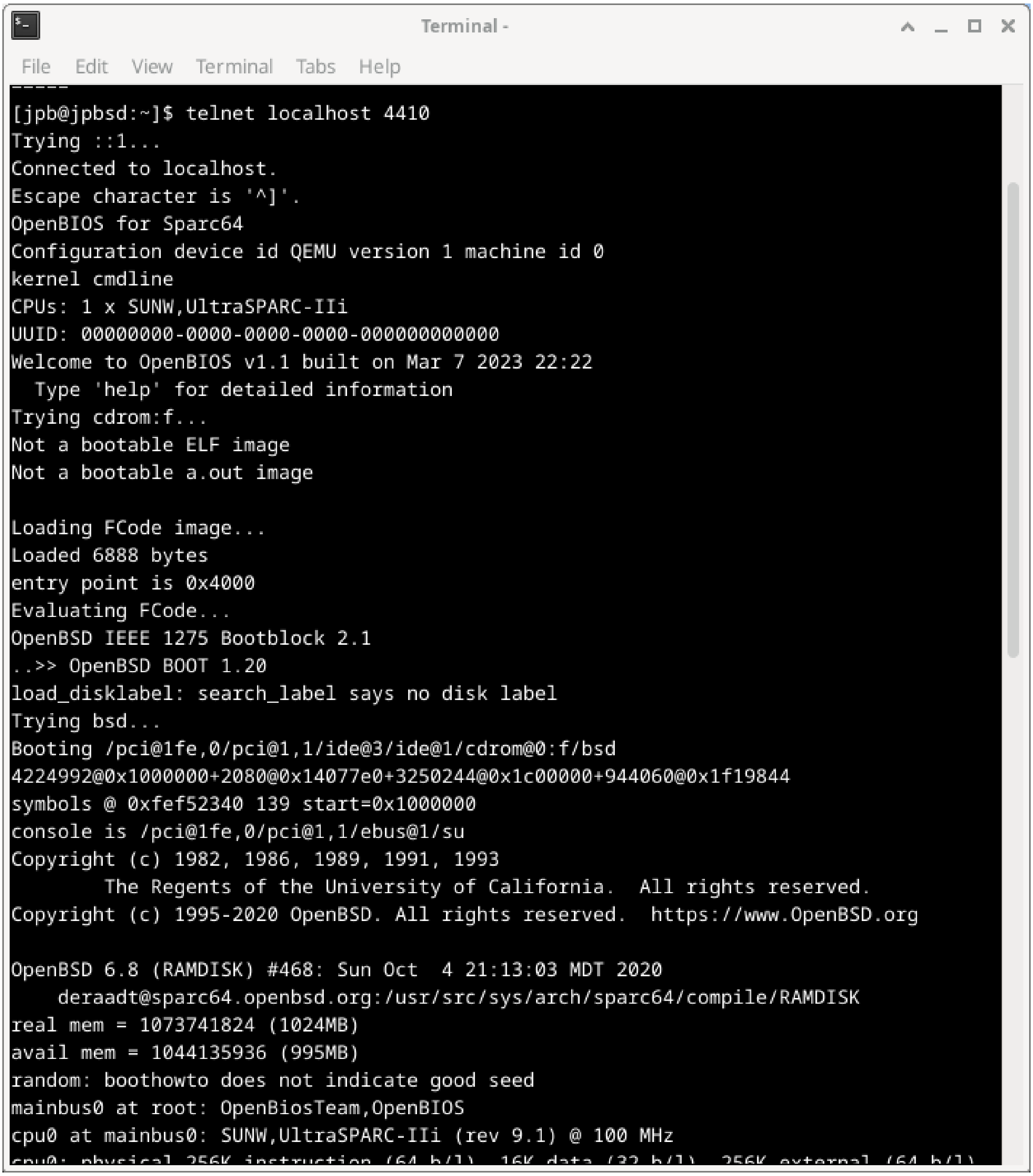
   ```
   echo
   echo "NOTE: telnet startup server running on guest VM!"
   echo "To start QEMU, start another session and telnet to localhost port 4410"
   echo

   /usr/local/bin/qemu-system-sparc64 \
     -serial telnet:localhost:4410,server=on,wait=on \
     -machine sun4u,usb=off \
   ```

```
  -smp 1,sockets=1,cores=1,threads=1 \
  -rtc base=utc \
  -m 1024 \
  -boot d \
  -drive file=../VM/sparc64.qcow2,if=none,id=drive-ide0-0-1,format=qcow2,cache=none \
  -cdrom ../ISO/install68.iso \
  -device ide-hd,bus=ide.0,unit=0,drive=drive-ide0-0-1,id=ide0-0-1 \
  -msg timestamp=on \
  -net nic,model=sunhme -net user \
  -nographic \
  -name \"sparc64\"
```

Note the following:

- The `-boot d` option boots from the QEMU CDROM device which is set as `-cdrom ../ISO/install68.iso`.
- As before, the `telnet` server option is set to wait for a separate connection on port 4410. Start up another session and use telnet(1) to connect to localhost on port 4410.
- The script sets the `-nographic` option meaning there is only serial port I/O. There is no graphical interface.
- Networking is not set up through the tap(4) / if_bridge(4) combination. This example uses a separate method of QEMU networking known as "Serial Line Internet Protocol" (SLIRP), sometimes referred to as "User Mode Networking". Documentation on this and other QEMU networking methods is here: QEMU Networking Documentation

If everything is set correctly, the system will boot as shown in Figure 9.



```
[jpb@jpbsd:~]$ telnet localhost 4410
Trying ::1...
Connected to localhost.
Escape character is '^]'.
OpenBIOS for Sparc64
Configuration device id QEMU version 1 machine id 0
kernel cmdline
CPUs: 1 x SUNW,UltraSPARC-IIi
UUID: 00000000-0000-0000-0000-000000000000
Welcome to OpenBIOS v1.1 built on Mar 7 2023 22:22
  Type 'help' for detailed information
Trying cdrom:f...
Not a bootable ELF image
Not a bootable a.out image

Loading FCode image...
Loaded 6888 bytes
entry point is 0x4000
Evaluating FCode...
OpenBSD IEEE 1275 Bootblock 2.1
..>> OpenBSD BOOT 1.20
load_disklabel: search_label says no disk label
Trying bsd...
Booting /pci@1fe,0/pci@1,1/ide@3/ide@1/cdrom@0:f/bsd
4224992@0x1000000+2080@0x14077e0+3250244@0x1c00000+944060@0x1f19844
symbols @ 0xfef52340 139 start=0x1000000
console is /pci@1fe,0/pci@1,1/ebus@1/su
Copyright (c) 1982, 1986, 1989, 1991, 1993
        The Regents of the University of California.  All rights reserved.
Copyright (c) 1995-2020 OpenBSD. All rights reserved.  https://www.OpenBSD.org

OpenBSD 6.8 (RAMDISK) #468: Sun Oct  4 21:13:03 MDT 2020
    deraadt@sparc64.openbsd.org:/usr/src/sys/arch/sparc64/compile/RAMDISK
real mem = 1073741824 (1024MB)
avail mem = 1044135936 (995MB)
random: boothowto does not indicate good seed
mainbus0 at root: OpenBiosTeam,OpenBIOS
cpu0 at mainbus0: SUNW,UltraSPARC-IIi (rev 9.1) @ 100 MHz
```

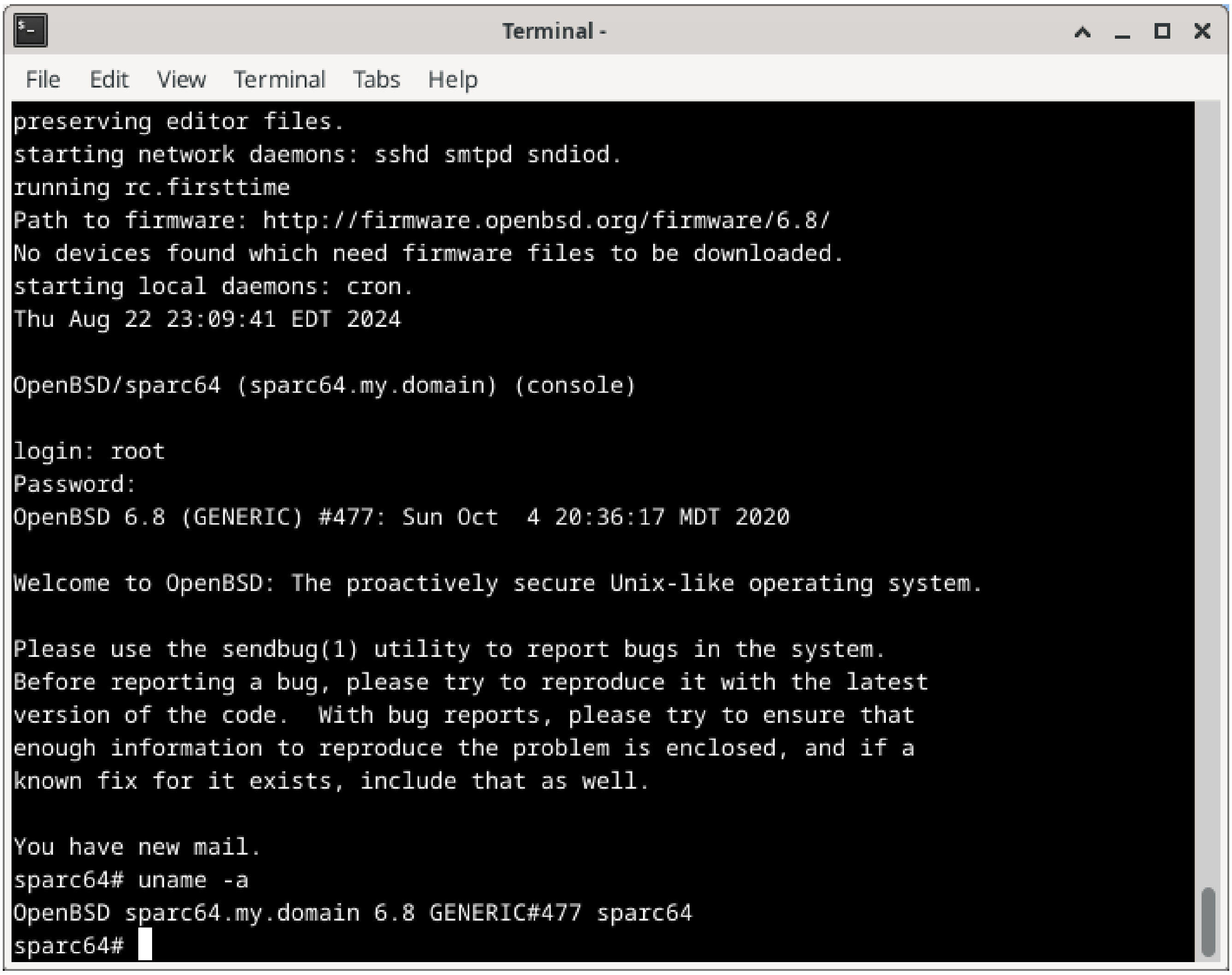
Figure 68. QEMU Booting OpenBSD 6.8 Sparc64 from CDROM During User Mode Emulation

Once the system is installed, modify the script and change the boot parameter to `-boot c`. This will indicate to QEMU to boot from the supplied hard disk, not the CDROM.

The installed system can be used just like any other guest virtual machine. However, the underlying architecture of the guest is Sparc64, not x86_64.

If the system is halted at the OpenBios console prompt `0 >`, enter `power-off` to exit the system.

Figure 10 shows a root login to the installed system and running uname(1).



```
preserving editor files.
starting network daemons: sshd smtpd sndiod.
running rc.firsttime
Path to firmware: http://firmware.openbsd.org/firmware/6.8/
No devices found which need firmware files to be downloaded.
starting local daemons: cron.
Thu Aug 22 23:09:41 EDT 2024

OpenBSD/sparc64 (sparc64.my.domain) (console)

login: root
Password:
OpenBSD 6.8 (GENERIC) #477: Sun Oct  4 20:36:17 MDT 2020

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code.  With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

You have new mail.
sparc64# uname -a
OpenBSD sparc64.my.domain 6.8 GENERIC#477 sparc64
sparc64#
```

Figure 69. QEMU Booting from CDROM During User Mode Emulation

25.6.6. Using the QEMU Monitor

The QEMU monitor controls a running QEMU emulator (guest VM).

Using the monitor, it is possible to:

- Dynamically remove or insert devices, including disks, network interfaces, CD-ROMs, or floppies
- Freeze/unfreeze the guest VM, and save or restore its state from a disk file
- Gather information about the state of the VM and devices
- Change device settings on the fly

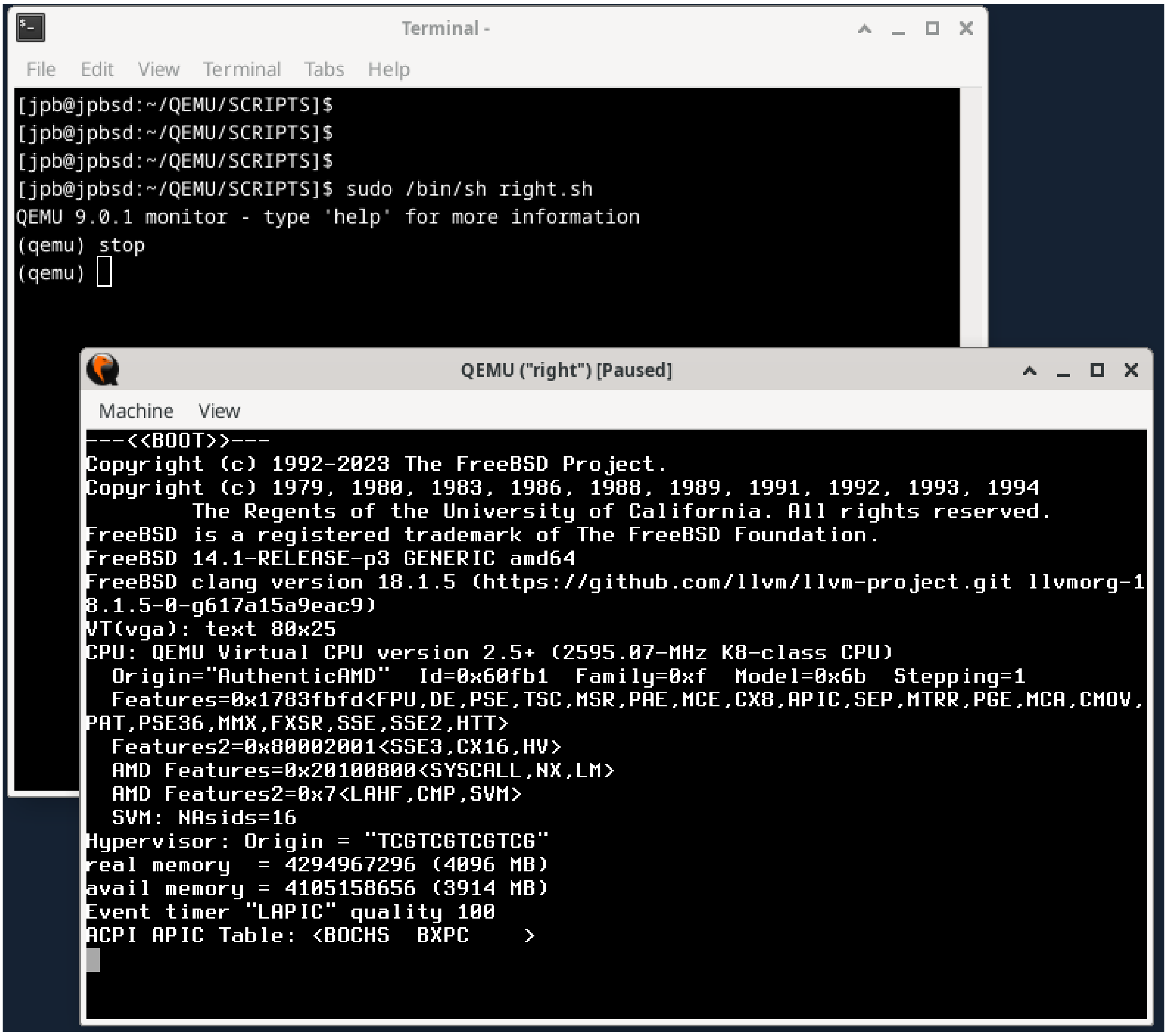
As well as many other operations.

The most common uses of the monitor are to examine the state of the VM, and to add, delete, or change devices. Some operations such as migrations are only available under hypervisor accelerators such as KVM, Xen, etc. and are not supported on FreeBSD hosts.

When using a graphical desktop environment, the simplest way to use the QEMU monitor is the `-monitor stdio` option when launching QEMU from a terminal session.

```
# /usr/local/bin/qemu-system-x86_64  -monitor stdio \
  -cpu qemu64 \
  -vga cirrus \
  -m 4096  -smp 4   \
  ...
```

This results in a new prompt `(qemu)` in the terminal window as shown in Figure 11.



```
[jpb@jpbsd:~/QEMU/SCRIPTS]$
[jpb@jpbsd:~/QEMU/SCRIPTS]$
[jpb@jpbsd:~/QEMU/SCRIPTS]$
[jpb@jpbsd:~/QEMU/SCRIPTS]$ sudo /bin/sh right.sh
QEMU 9.0.1 monitor - type 'help' for more information
(qemu) stop
(qemu) 
```

```
---<<BOOT>>---
Copyright (c) 1992-2023 The FreeBSD Project.
Copyright (c) 1979, 1980, 1983, 1986, 1988, 1989, 1991, 1992, 1993, 1994
        The Regents of the University of California. All rights reserved.
FreeBSD is a registered trademark of The FreeBSD Foundation.
FreeBSD 14.1-RELEASE-p3 GENERIC amd64
FreeBSD clang version 18.1.5 (https://github.com/llvm/llvm-project.git llvmorg-1
8.1.5-0-g617a15a9eac9)
VT(vga): text 80x25
CPU: QEMU Virtual CPU version 2.5+ (2595.07-MHz K8-class CPU)
  Origin="AuthenticAMD"  Id=0x60fb1  Family=0xf  Model=0x6b  Stepping=1
  Features=0x1783fbfd<FPU,DE,PSE,TSC,MSR,PAE,MCE,CX8,APIC,SEP,MTRR,PGE,MCA,CMOV,
PAT,PSE36,MMX,FXSR,SSE,SSE2,HTT>
  Features2=0x80002001<SSE3,CX16,HV>
  AMD Features=0x20100800<SYSCALL,NX,LM>
  AMD Features2=0x7<LAHF,CMP,SVM>
  SVM: NAsids=16
Hypervisor: Origin = "TCGTCGTCGTCG"
real memory  = 4294967296 (4096 MB)
avail memory = 4105158656 (3914 MB)
Event timer "LAPIC" quality 100
ACPI APIC Table: <BOCHS  BXPC    >
```

Figure 70. QEMU Monitor Prompt and "stop" Command

The image also shows the `stop` command freezing the system during the FreeBSD boot sequence. The system will remain frozen until the `cont` command is entered in the monitor.

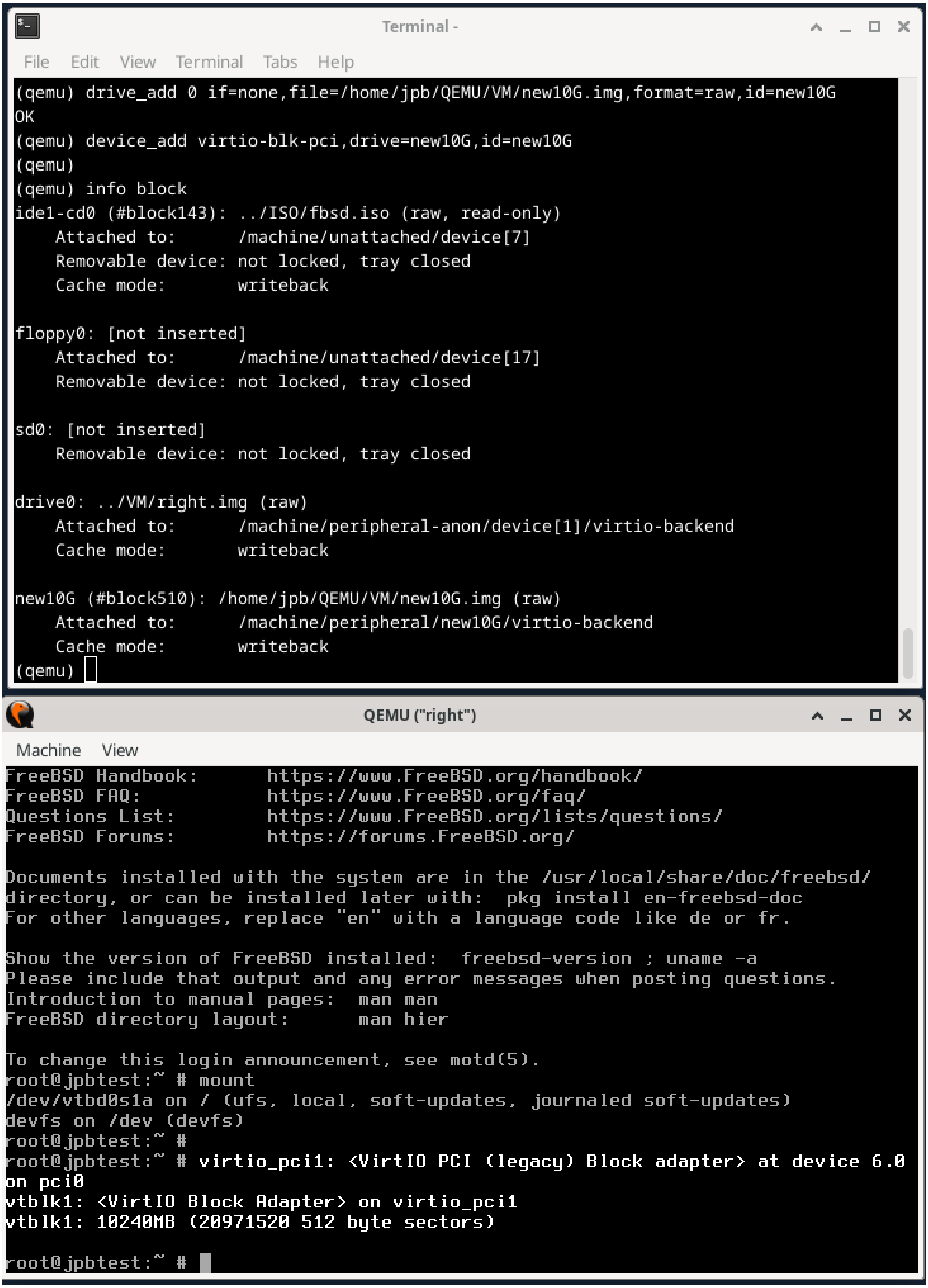
Adding a New Disk to the VM

To add a new disk to a running VM, the disk needs to be prepared as above:

```
% cd ~/QEMU/VM
% qemu-img create -f raw  new10G.img  10G
```

Figure 12 shows the monitor command sequence needed to add a new disk in the VM. Once the device has been added with the `device_add` command in the monitor it shows up on the FreeBSD system console shown in the lower part of the figure. The disk can be configured as needed.

Note that the new disk must be added to the startup script if it is to be used after a VM reboot.



```
(qemu) drive_add 0 if=none,file=/home/jpb/QEMU/VM/new10G.img,format=raw,id=new10G
OK
(qemu) device_add virtio-blk-pci,drive=new10G,id=new10G
(qemu)
(qemu) info block
ide1-cd0 (#block143): ../ISO/fbsd.iso (raw, read-only)
    Attached to:      /machine/unattached/device[7]
    Removable device: not locked, tray closed
    Cache mode:       writeback

floppy0: [not inserted]
    Attached to:      /machine/unattached/device[17]
    Removable device: not locked, tray closed

sd0: [not inserted]
    Removable device: not locked, tray closed

drive0: ../VM/right.img (raw)
    Attached to:      /machine/peripheral-anon/device[1]/virtio-backend
    Cache mode:       writeback

new10G (#block510): /home/jpb/QEMU/VM/new10G.img (raw)
    Attached to:      /machine/peripheral/new10G/virtio-backend
    Cache mode:       writeback
(qemu)
```

```
FreeBSD Handbook:         https://www.FreeBSD.org/handbook/
FreeBSD FAQ:              https://www.FreeBSD.org/faq/
Questions List:           https://www.FreeBSD.org/lists/questions/
FreeBSD Forums:           https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
FreeBSD directory layout:      man hier

To change this login announcement, see motd(5).
root@jpbtest:~ # mount
/dev/vtbd0s1a on / (ufs, local, soft-updates, journaled soft-updates)
devfs on /dev (devfs)
root@jpbtest:~ #
root@jpbtest:~ # virtio_pci1: <VirtIO PCI (legacy) Block adapter> at device 6.0
on pci0
vtblk1: <VirtIO Block Adapter> on virtio_pci1
vtblk1: 10240MB (20971520 512 byte sectors)

root@jpbtest:~ #
```

Figure 71. QEMU Monitor Commands to Add a New Disk

Using the QEMU Monitor to Manage Snapshots

QEMU's documentation describes several similar concepts when using the term **snapshot**. There is the `-snapshot` option on the command line which refers to using a drive or portion of a drive to contain a copy of a device. Then there are the monitor commands `snapshot_blkdev` and `snapshot_blkdev_internal` which describe the actual act of copying the blockdev device. Finally, there are the monitor commands `savevm`, `loadvm`, and `delvm` commands which refer to creating and saving, loading, or deleting a copy of an entire virtual machine. Along with the latter, the monitor `info snapshots` command lists out details of recent snapshots.

This section will focus on creating, saving, and loading a complete VM image and will use the term **snapshot** for this purpose.

To start, recreate the "left" VM from scratch, this time using the `qcow2` format.

```
% cd ~/QEMU/VM
% rm left.img
% qemu-img create -f qcow2 left.qcow2 16G  # Clean file for a new FreeBSD
installation.
% cd ../SCRIPTS
# /bin/sh left.sh                        # See the below program listing.
```

Once the installation is complete, reboot, this time using the `-monitor stdio` option to allow use of the monitor.

```
# left VM script.
/usr/local/bin/qemu-system-x86_64  -monitor stdio \
  -cpu qemu64 \
  -vga std \
  -m 4096 \
  -smp 4   \
  -cdrom ../ISO/fbsd.iso \
  -boot order=cd,menu=on \
  -blockdev driver=file,aio=threads,node-name=imgleft,filename=../VM/left.qcow2 \
  -blockdev driver=qcow2,node-name=drive0,file=imgleft \
  -device virtio-blk-pci,drive=drive0,bootindex=1  \
  -netdev tap,id=nd0,ifname=tap0,script=no,downscript=no,br=bridge0 \
  -device e1000,netdev=nd0,mac=02:20:6c:65:66:74 \
  -name \"left\"
```

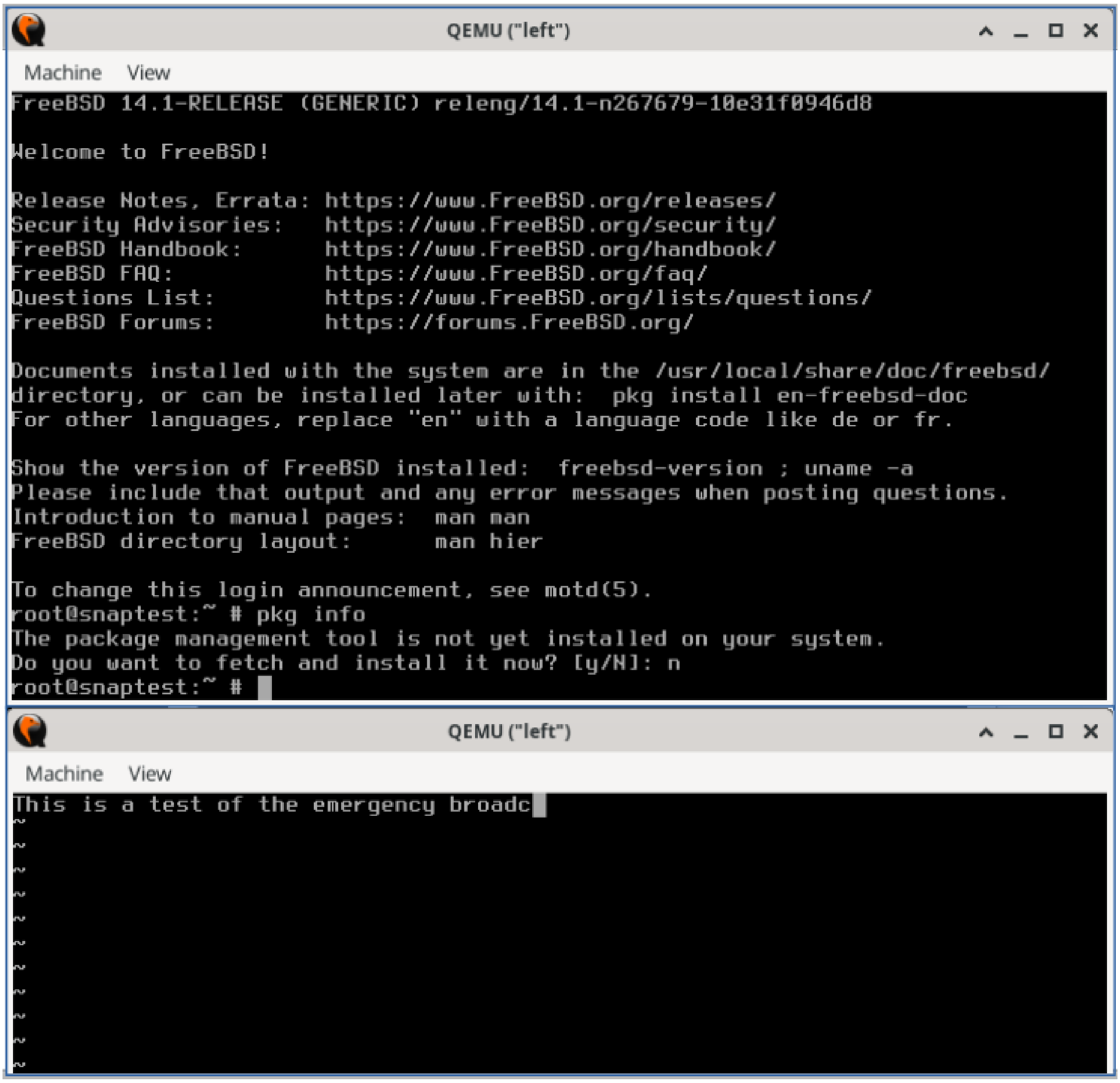
To demonstrate snapshots, the following procedure can be used:

1. Install FreeBSD from scratch
2. Prepare the environment and take a snapshot with the `savevm` monitor command
3. Install several packages
4. Shut down the system
5. Restart a bare QEMU instance and utilize the monitor command `loadvm` to restore the VM

6. Observe that the restored VM does not have any packages

During the "Prepare the environment" step, in a separate virtual console (ttyv1), an editing session with vi(1) is initiated simulating user activity. Additional programs may be started if desired. The snapshot should account for the state of all applications running at the time the snapshot is taken.

Figure 13 shows the newly installed FreeBSD system with no packages, and separately, the editing session on ttyv1. The vi(1) editor is currently in `insert` mode with the typist typing the word "broadcast".



```
FreeBSD 14.1-RELEASE (GENERIC) releng/14.1-n267679-10e31f0946d8

Welcome to FreeBSD!

Release Notes, Errata: https://www.FreeBSD.org/releases/
Security Advisories:   https://www.FreeBSD.org/security/
FreeBSD Handbook:      https://www.FreeBSD.org/handbook/
FreeBSD FAQ:           https://www.FreeBSD.org/faq/
Questions List:        https://www.FreeBSD.org/lists/questions/
FreeBSD Forums:        https://forums.FreeBSD.org/

Documents installed with the system are in the /usr/local/share/doc/freebsd/
directory, or can be installed later with:  pkg install en-freebsd-doc
For other languages, replace "en" with a language code like de or fr.

Show the version of FreeBSD installed:  freebsd-version ; uname -a
Please include that output and any error messages when posting questions.
Introduction to manual pages:  man man
FreeBSD directory layout:      man hier

To change this login announcement, see motd(5).
root@snaptest:~ # pkg info
The package management tool is not yet installed on your system.
Do you want to fetch and install it now? [y/N]: n
root@snaptest:~ #
```

```
This is a test of the emergency broadc
~
~
~
~
~
~
~
~
~
~
~
```

Figure 72. QEMU VM Before First Snapshot

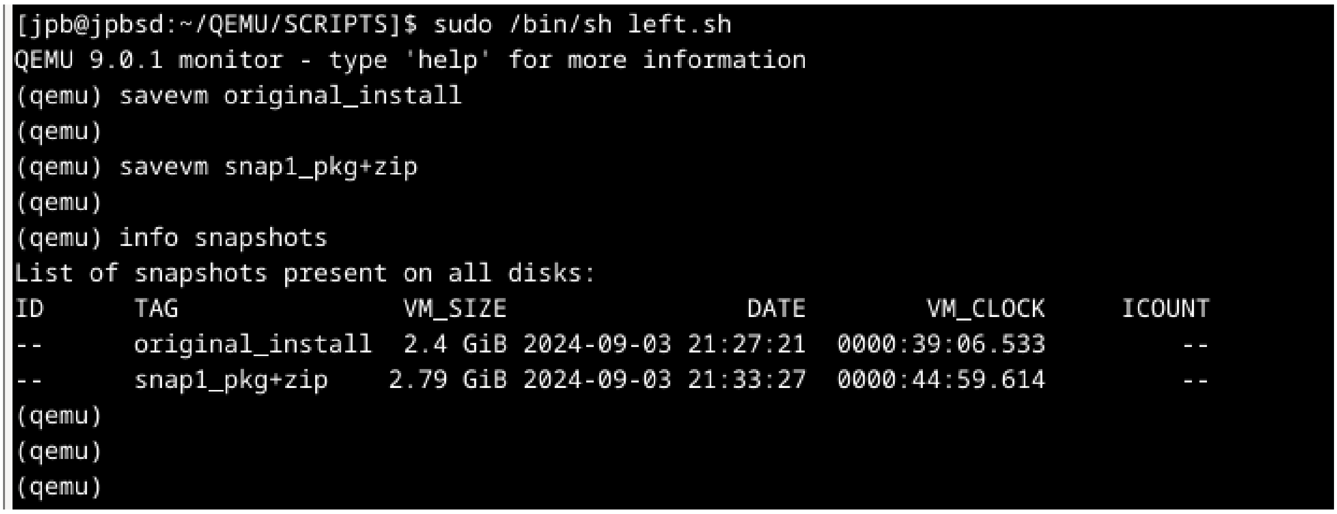
To generate the snapshot, enter `savevm` in the monitor. Be sure to give it a tag (such as `original_install`).

```
QEMU 9.0.1 monitor - type 'help' for more information
(qemu)
```

```
(qemu) savevm original_install
```

Next, in the main console window, install a package, such as zip(1) which has no dependencies. Once that completes, renter the monitor and create another snapshot (`snap1_pkg+zip`).

Figure 14 shows the results of the above commands and the output of the `info shapshots` command.



```
[jpb@jpbsd:~/QEMU/SCRIPTS]$ sudo /bin/sh left.sh
QEMU 9.0.1 monitor - type 'help' for more information
(qemu) savevm original_install
(qemu)
(qemu) savevm snap1_pkg+zip
(qemu)
(qemu) info snapshots
List of snapshots present on all disks:
ID        TAG                 VM_SIZE                DATE        VM_CLOCK     ICOUNT
--        original_install   2.4 GiB 2024-09-03 21:27:21   0000:39:06.533         --
--        snap1_pkg+zip     2.79 GiB 2024-09-03 21:33:27   0000:44:59.614         --
(qemu)
(qemu)
(qemu)
```

Figure 73. QEMU Using Monitor Commands for Snapshots

Reboot the system, and before FreeBSD starts up, switch to the monitor and enter `stop`. The VM will stop.

Enter `loadvm` with the tag used above (here `original_install`).

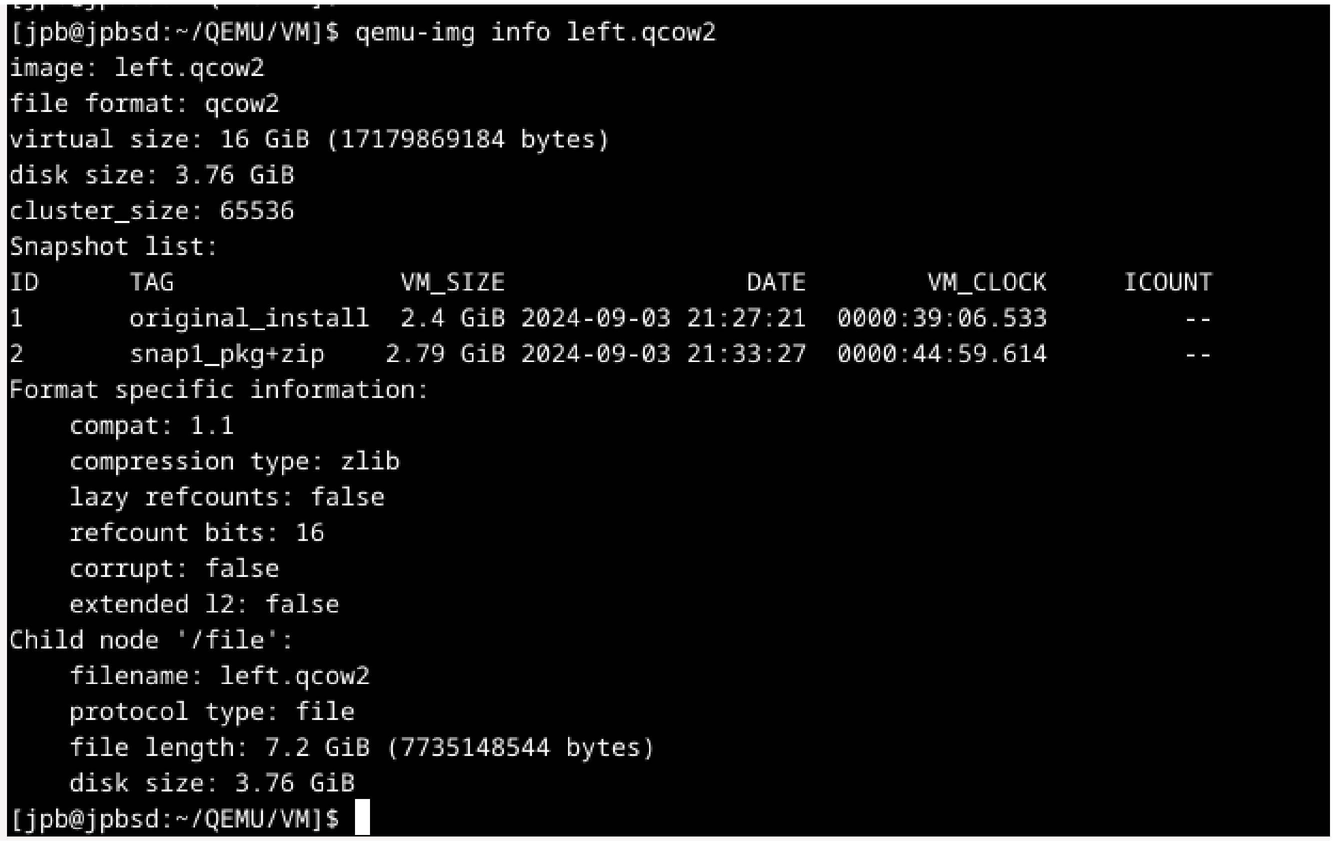
```
QEMU 9.0.1 monitor - type 'help' for more information
(qemu) stop
(qemu) loadvm original_install
(qemu) cont
```

Immediately, the VM screen will switch to the exact moment the `savevm` command was entered above. Note that the VM is still stopped.

Enter `cont` to start the VM, switch to the editing session on `ttyv1`, and type one letter on the keyboard. The editor, still in insert mode, should respond accordingly. Any other programs running at the time the snapshot was taken should be unaffected.

The above steps show how a snapshot can be taken, the system modified, and then "rolled back" by restoring the previous snapshot.

By default QEMU stores snapshot data in the same file as the image. View the list of snapshots with qemu-img(1) as shown below in Figure 15.



```
[jpb@jpbsd:~/QEMU/VM]$ qemu-img info left.qcow2
image: left.qcow2
file format: qcow2
virtual size: 16 GiB (17179869184 bytes)
disk size: 3.76 GiB
cluster_size: 65536
Snapshot list:
ID        TAG                 VM_SIZE                DATE        VM_CLOCK     ICOUNT
1         original_install   2.4 GiB 2024-09-03 21:27:21   0000:39:06.533        --
2         snap1_pkg+zip     2.79 GiB 2024-09-03 21:33:27   0000:44:59.614        --
Format specific information:
    compat: 1.1
    compression type: zlib
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
    extended l2: false
Child node '/file':
    filename: left.qcow2
    protocol type: file
    file length: 7.2 GiB (7735148544 bytes)
    disk size: 3.76 GiB
[jpb@jpbsd:~/QEMU/VM]$
```

Figure 74. QEMU Using qemu-img(1) to Examine Snapshots

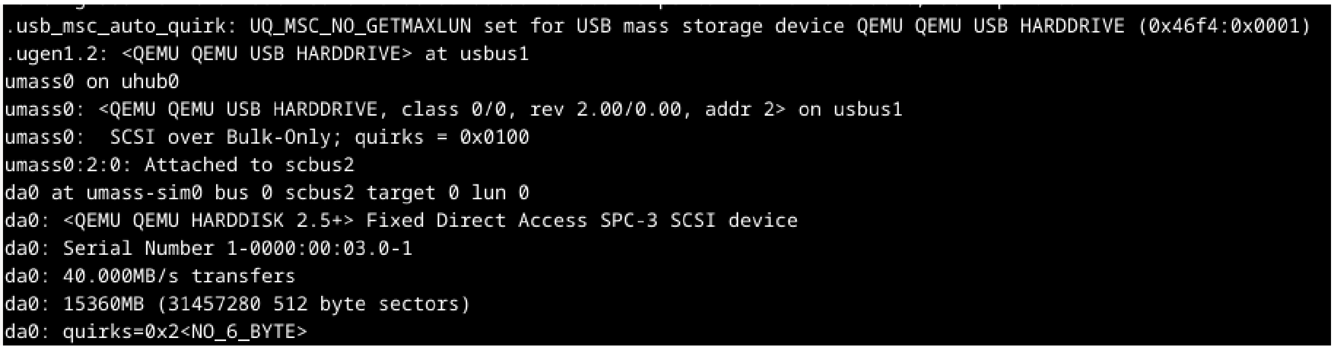
25.6.7. Using QEMU USB Devices

QEMU supports the creation of virtual USB devices that are backed by an image file. These are virtual USB devices that can be partitioned, formatted, mounted, and used just like a real USB device.

```
/usr/local/bin/qemu-system-x86_64  -monitor stdio \
  -cpu qemu64 \
  -vga cirrus \
  -m 4096  -smp 4   \
  -cdrom ../ISO/fbsd.iso \
  -boot order=cd,menu=on \
  -drive if=none,id=usbstick,format=raw,file=../VM/foo.img \
  -usb \
  -device usb-ehci,id=ehci \
  -device usb-storage,bus=ehci.0,drive=usbstick \
  -device usb-mouse \
  -blockdev driver=file,node-name=img1,filename=../VM/right.qcow2 \
  -blockdev driver=qcow2,node-name=drive0,file=img1 \
  -device virtio-blk-pci,drive=drive0,bootindex=1  \
  -netdev tap,id=nd0,ifname=tap1,script=no,downscript=no,br=bridge0 \
  -device e1000,netdev=nd0,mac=02:72:69:67:68:74 \
  -name \"right\"
```

This configuration includes a `-drive` specification with the `id=usbstick`, raw format, and an image file (must be created by qemu-img(1)). The next line contains the `-device usb-ehci` specification for a USB EHCI controller, with `id=ehci`. Finally, a `-device usb-storage` specification ties the above drive to the EHCI USB bus.

When the system is booted, FreeBSD will recognize a USB hub, add the attached USB device, and assign it to `da0` as shown in Figure 16.



```
.usb_msc_auto_quirk: UQ_MSC_NO_GETMAXLUN set for USB mass storage device QEMU QEMU USB HARDDRIVE (0x46f4:0x0001)
.ugen1.2: <QEMU QEMU USB HARDDRIVE> at usbus1
umass0 on uhub0
umass0: <QEMU QEMU USB HARDDRIVE, class 0/0, rev 2.00/0.00, addr 2> on usbus1
umass0:  SCSI over Bulk-Only; quirks = 0x0100
umass0:2:0: Attached to scbus2
da0 at umass-sim0 bus 0 scbus2 target 0 lun 0
da0: <QEMU QEMU HARDDISK 2.5+> Fixed Direct Access SPC-3 SCSI device
da0: Serial Number 1-0000:00:03.0-1
da0: 40.000MB/s transfers
da0: 15360MB (31457280 512 byte sectors)
da0: quirks=0x2<NO_6_BYTE>
```

Figure 75. QEMU Created USB Hub and Mass Storage Device

The device is ready to be partitioned with gpart(8), and formatted with newfs(8). Because the USB device is backed by a qemu-img(1) created file, data written to the device will persist across reboots.

25.6.8. Using Host USB Devices via Passthrough

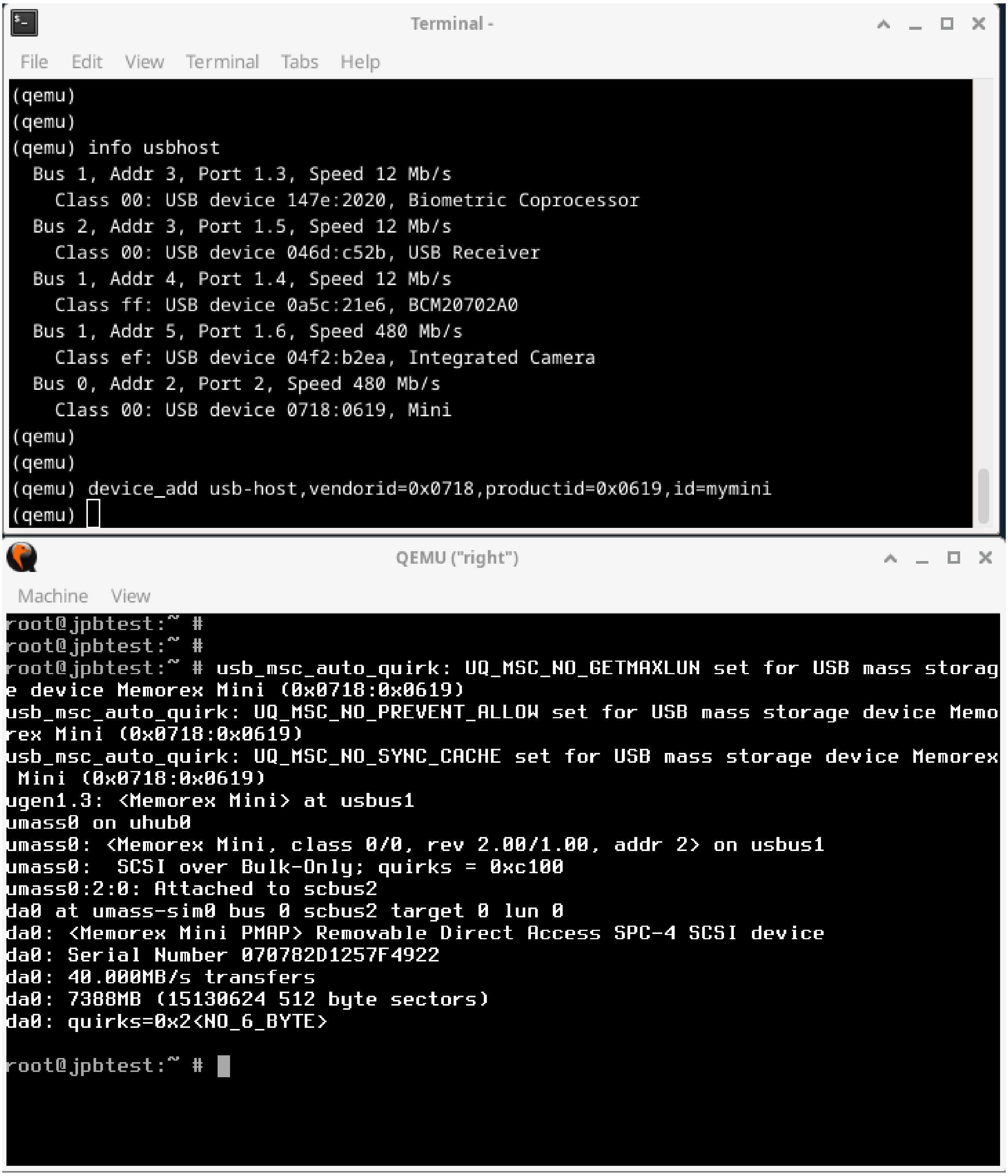
QEMU USB passthrough support is listed as experimental in version 9.0.1 (Summer, 2024). However, the following steps show how a USB stick mounted on the host can be used by the guest VM.

For more information and examples, see:

- https://www.qemu.org/docs/master/system/devices/usb.html

The upper part of Figure 17 shows the QEMU monitor commands:

- `info usbhost` shows information about all USB devices on the host system. Find the desired USB device on the host system and note the two hexadecimal values on that line. (In the example below the host USB device is a Memorex Mini, with vendorid 0718, and productid 0619.) Use the two values shown by the `info usbhost` command in the `device_add` step below.
- `device_add` adds a USB device to the guest VM.



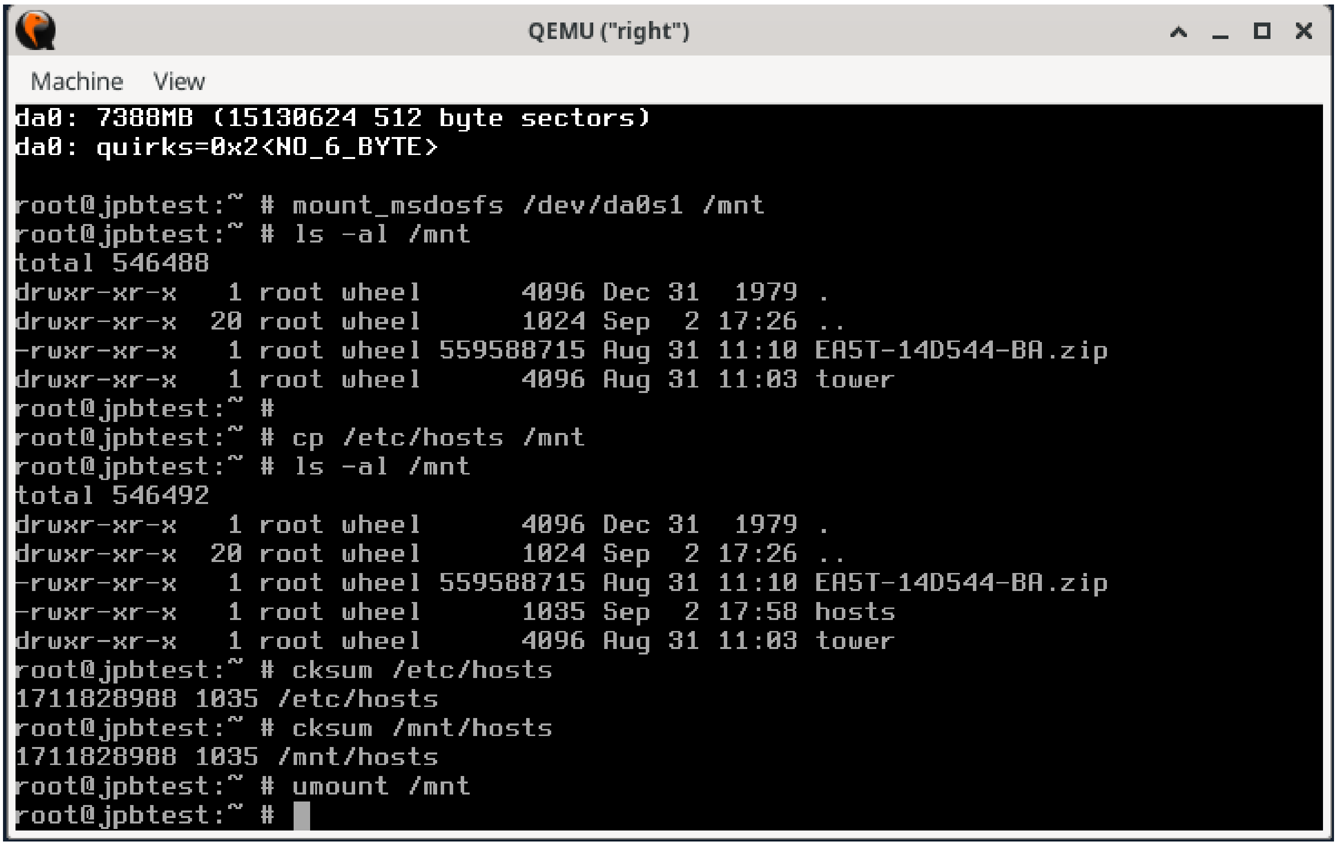
```
(qemu)
(qemu)
(qemu) info usbhost
  Bus 1, Addr 3, Port 1.3, Speed 12 Mb/s
    Class 00: USB device 147e:2020, Biometric Coprocessor
  Bus 2, Addr 3, Port 1.5, Speed 12 Mb/s
    Class 00: USB device 046d:c52b, USB Receiver
  Bus 1, Addr 4, Port 1.4, Speed 12 Mb/s
    Class ff: USB device 0a5c:21e6, BCM20702A0
  Bus 1, Addr 5, Port 1.6, Speed 480 Mb/s
    Class ef: USB device 04f2:b2ea, Integrated Camera
  Bus 0, Addr 2, Port 2, Speed 480 Mb/s
    Class 00: USB device 0718:0619, Mini
(qemu)
(qemu)
(qemu) device_add usb-host,vendorid=0x0718,productid=0x0619,id=mymini
(qemu) 
```

```
root@jpbtest:~ #
root@jpbtest:~ #
root@jpbtest:~ # usb_msc_auto_quirk: UQ_MSC_NO_GETMAXLUN set for USB mass storag
e device Memorex Mini (0x0718:0x0619)
usb_msc_auto_quirk: UQ_MSC_NO_PREVENT_ALLOW set for USB mass storage device Memo
rex Mini (0x0718:0x0619)
usb_msc_auto_quirk: UQ_MSC_NO_SYNC_CACHE set for USB mass storage device Memorex
 Mini (0x0718:0x0619)
ugen1.3: <Memorex Mini> at usbus1
umass0 on uhub0
umass0: <Memorex Mini, class 0/0, rev 2.00/1.00, addr 2> on usbus1
umass0:  SCSI over Bulk-Only; quirks = 0xc100
umass0:2:0: Attached to scbus2
da0 at umass-sim0 bus 0 scbus2 target 0 lun 0
da0: <Memorex Mini PMAP> Removable Direct Access SPC-4 SCSI device
da0: Serial Number 070782D1257F4922
da0: 40.000MB/s transfers
da0: 7388MB (15130624 512 byte sectors)
da0: quirks=0x2<NO_6_BYTE>

root@jpbtest:~ # 
```

Figure 76. QEMU Monitor Commands to Access a USB Device on the Host

As before, once `device_add` completes, the FreeBSD kernel recognizes a new USB device, as shown in the lower half of the .

Using the new device is shown in Figure 18.



```
da0: 7388MB (15130624 512 byte sectors)
da0: quirks=0x2<NO_6_BYTE>

root@jpbtest:~ # mount_msdosfs /dev/da0s1 /mnt
root@jpbtest:~ # ls -al /mnt
total 546488
drwxr-xr-x   1 root wheel       4096 Dec 31  1979 .
drwxr-xr-x  20 root wheel       1024 Sep  2 17:26 ..
-rwxr-xr-x   1 root wheel 559588715 Aug 31 11:10 EA5T-14D544-BA.zip
drwxr-xr-x   1 root wheel       4096 Aug 31 11:03 tower
root@jpbtest:~ #
root@jpbtest:~ # cp /etc/hosts /mnt
root@jpbtest:~ # ls -al /mnt
total 546492
drwxr-xr-x   1 root wheel       4096 Dec 31  1979 .
drwxr-xr-x  20 root wheel       1024 Sep  2 17:26 ..
-rwxr-xr-x   1 root wheel 559588715 Aug 31 11:10 EA5T-14D544-BA.zip
-rwxr-xr-x   1 root wheel       1035 Sep  2 17:58 hosts
drwxr-xr-x   1 root wheel       4096 Aug 31 11:03 tower
root@jpbtest:~ # cksum /etc/hosts
1711828988 1035 /etc/hosts
root@jpbtest:~ # cksum /mnt/hosts
1711828988 1035 /mnt/hosts
root@jpbtest:~ # umount /mnt
root@jpbtest:~ #
```

Figure 77. Using the Host USB Device via Passthrough

If the USB device is formatted as a FAT16 or FAT32 filesystem it can be mounted as an MS-DOS™ filesystem with mount_msdosfs(8) as in the example shown. The `/etc/hosts` file is copied to the newly mounted drive and checksums are taken to verify the integrity of the file on the USB device. The device is then unmounted with umount(8).

If the USB device is formatted with NTFS it is necessary to install the `fusefs-ntfs` package and use ntfs-3g(8) to access the device:

```
# pkg install fusefs-ntfs
# kldload fusefs
# gpart show da1
# ntfs-3g /dev/da1s1 /mnt

Access the drive as needed.  When finished:

# umount /mnt
```

Change the above device identifiers to match the installed hardware. Consult ntfs-3g(8) for additional information on working with NTFS filesystems.

25.6.9. QEMU on FreeBSD Summary

As noted above, QEMU works with several different hypervisor accelerators.

The list of Virtualization Accelerators supported by QEMU includes:

- `KVM` on Linux supporting 64 bit Arm, MIPS, PPC, RISC-V, s390x, and x86
- `Xen` on Linux as dom0 supporting Arm, x86
- `Hypervisor Framework (hvf)` on MacOS supporting x86 and Arm (both 64 bit only)
- `Windows Hypervisor Platform (whpx)` on Windows supporting x86
- `NetBSD Virtual Machine Monitor (nvmm)` on NetBSD supporting x86
- `Tiny Code Generator (tcg)` on Linux and other POSIX, Windows, MacOS supporting Arm, x86, Loongarch64, MIPS, PPC, s390x, and Sparc64.

All the examples in this section used the `Tiny Code Generator (tcg)` accelerator as that is the only supported accelerator on FreeBSD at present.

25.7. FreeBSD as a Host with bhyve

The bhyve BSD-licensed hypervisor became part of the base system with FreeBSD 10.0-RELEASE. This hypervisor supports several guests, including FreeBSD, OpenBSD, many Linux® distributions, and Microsoft Windows®. By default, bhyve provides access to a serial console and does not emulate a graphical console. Virtualization offload features of newer CPUs are used to avoid the legacy methods of translating instructions and manually managing memory mappings.

The bhyve design requires

- an Intel® processor that supports Intel Extended Page Tables (EPT),
- or an AMD® processor that supports AMD Rapid Virtualization Indexing (RVI), or Nested Page Tables (NPT),
- or an ARM® aarch64 CPU.

Only pure ARMv8.0 virtualization is supported on ARM, the Virtualization Host Extensions are not currently used. Hosting Linux® guests or FreeBSD guests with more than one vCPU requires VMX unrestricted mode support (UG).

The easiest way to tell if an Intel or AMD processor supports bhyve is to run `dmesg` or look in /var/run/dmesg.boot for the `POPCNT` processor feature flag on the `Features2` line for AMD® processors or `EPT` and `UG` on the `VT-x` line for Intel® processors.

25.7.1. Preparing the Host

The first step to creating a virtual machine in bhyve is configuring the host system. First, load the bhyve kernel module:

```
# kldload vmm
```

There are several ways to connect a virtual machine guest to a host's network; one straightforward way to accomplish this is to create a tap interface for the network device in the virtual machine to attach to. For the network device to participate in the network, also create a bridge interface containing the tap interface and the physical interface as members. In this example, the physical interface is *igb0*:

```
# ifconfig tap0 create
# sysctl net.link.tap.up_on_open=1
net.link.tap.up_on_open: 0 -> 1
# ifconfig bridge0 create
# ifconfig bridge0 addm igb0 addm tap0
# ifconfig bridge0 up
```

25.7.2. Creating a FreeBSD Guest

Create a file to use as the virtual disk for the guest machine. Specify the size and name of the virtual disk:

```
# truncate -s 16G guest.img
```

Download an installation image of FreeBSD to install:

```
# fetch https://download.freebsd.org/releases/ISO-IMAGES/14.0/FreeBSD-14.0-RELEASE-
amd64-bootonly.iso
FreeBSD-14.0-RELEASE-amd64-bootonly.iso                426 MB   16 MBps    22s
```

FreeBSD comes with an example script `vmrun.sh` for running a virtual machine in bhyve. It will start the virtual machine and run it in a loop, so it will automatically restart if it crashes. `vmrun.sh` takes several options to control the configuration of the machine, including:

- `-c` controls the number of virtual CPUs,
- `-m` limits the amount of memory available to the guest,
- `-t` defines which tap device to use,
- `-d` indicates which disk image to use,
- `-i` tells bhyve to boot from the CD image instead of the disk, and
- `-I` defines which CD image to use.

The last parameter is the name of the virtual machine and is used to track the running machines. The following command lists all available program argument options:

```
# sh /usr/share/examples/bhyve/vmrun.sh -h
```

This example starts the virtual machine in installation mode:

```
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M -t tap0 -d guest.img \
     -i -I FreeBSD-14.0-RELEASE-amd64-bootonly.iso guestname
```

The virtual machine will boot and start the installer. After installing a system in the virtual machine, when the system asks about dropping into a shell at the end of the installation, choose **[Yes]**.

Reboot the virtual machine. While rebooting the virtual machine causes bhyve to exit, the vmrun.sh script runs `bhyve` in a loop and will automatically restart it. When this happens, choose the reboot option from the boot loader menu to escape the loop. Now the guest can be started from the virtual disk:

```
# sh /usr/share/examples/bhyve/vmrun.sh -c 4 -m 1024M -t tap0 -d guest.img guestname
```

25.7.3. Creating a Linux® Guest

Linux guests can be booted either like any other regular UEFI-based guest virtual machine, or alternatively, use the sysutils/grub2-bhyve port.

To do this, first ensure that the port is installed, then create a file to use as the virtual disk for the guest machine:

```
# truncate -s 16G linux.img
```

Starting a Linux virtual machine with `grub2-bhyve` is a two-step process.

1. First a kernel must be loaded, then the guest can be started.
2. The Linux® kernel is loaded with sysutils/grub2-bhyve.

Create a device.map that grub will use to map the virtual devices to the files on the host system:

```
(hd0) ./linux.img
(cd0) ./somelinux.iso
```

Use sysutils/grub2-bhyve to load the Linux® kernel from the ISO image:

```
# grub-bhyve -m device.map -r cd0 -M 1024M linuxguest
```

This will start grub. If the installation CD contains a grub.cfg, a menu will be displayed. If not, the `vmlinuz` and `initrd` files must be located and loaded manually:

```
grub> ls
(hd0) (cd0) (cd0,msdos1) (host)
grub> ls (cd0)/isolinux
boot.cat boot.msg grub.conf initrd.img isolinux.bin isolinux.cfg memtest
splash.jpg TRANS.TBL vesamenu.c32 vmlinuz
grub> linux (cd0)/isolinux/vmlinuz
grub> initrd (cd0)/isolinux/initrd.img
grub> boot
```

Now that the Linux® kernel is loaded, the guest can be started:

```
# bhyve -A -H -P -s 0:0,hostbridge -s 1:0,lpc -s 2:0,virtio-net,tap0 \
    -s 3:0,virtio-blk,./linux.img -s 4:0,ahci-cd,./somelinux.iso \
    -l com1,stdio -c 4 -m 1024M linuxguest
```

The system will boot and start the installer. After installing a system in the virtual machine, reboot the virtual machine. This will cause bhyve to exit. The instance of the virtual machine needs to be destroyed before it can be started again:

```
# bhyvectl --destroy --vm=linuxguest
```

Now the guest can be started directly from the virtual disk. Load the kernel:

```
# grub-bhyve -m device.map -r hd0,msdos1 -M 1024M linuxguest
grub> ls
(hd0) (hd0,msdos2) (hd0,msdos1) (cd0) (cd0,msdos1) (host)
(lvm/VolGroup-lv_swap) (lvm/VolGroup-lv_root)
grub> ls (hd0,msdos1)/
lost+found/ grub/ efi/ System.map-2.6.32-431.el6.x86_64 config-2.6.32-431.el6.x
86_64 symvers-2.6.32-431.el6.x86_64.gz vmlinuz-2.6.32-431.el6.x86_64
initramfs-2.6.32-431.el6.x86_64.img
grub> linux (hd0,msdos1)/vmlinuz-2.6.32-431.el6.x86_64 root=/dev/mapper/VolGroup-
lv_root
grub> initrd (hd0,msdos1)/initramfs-2.6.32-431.el6.x86_64.img
grub> boot
```

Boot the virtual machine:

```
# bhyve -A -H -P -s 0:0,hostbridge -s 1:0,lpc -s 2:0,virtio-net,tap0 \
    -s 3:0,virtio-blk,./linux.img -l com1,stdio -c 4 -m 1024M linuxguest
```

Linux® will now boot in the virtual machine and eventually presents the login prompt. Login and use the virtual machine. When finished, reboot the virtual machine to exit bhyve. Destroy the virtual machine instance:

```
# bhyvectl --destroy --vm=linuxguest
```

25.7.4. Booting bhyve Virtual Machines with UEFI Firmware

In addition to `bhyveload` and `grub-bhyve`, the bhyve hypervisor can also boot virtual machines using the UEFI firmware. This option may support guest operating systems that are not supported by the other loaders.

To make use of the UEFI support in bhyve, first obtain the UEFI firmware images. This can be done by installing sysutils/bhyve-firmware port or package.

With the firmware in place, add the flags `-l bootrom,/path/to/firmware` to the bhyve command line. The actual bhyve command may look like this:

```
# bhyve -AHP -s 0:0,hostbridge -s 1:0,lpc \
    -s 2:0,virtio-net,tap1 -s 3:0,virtio-blk,./disk.img \
    -s 4:0,ahci-cd,./install.iso -c 4 -m 1024M \
    -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd \
    guest
```

To allow a guest to store UEFI variables, use a variables file appended to the `-l` flag. Note that bhyve will write guest modifications to the given variables file. Therefore, be sure to first create a per-guest-copy of the variables template file:

```
# cp /usr/local/share/uefi-firmware/BHYVE_UEFI_VARS.fd /path/to/vm-
image/BHYVE_UEFI_VARS.fd
```

Then, add that variables file to the bhyve arguments:

```
# bhyve -AHP -s 0:0,hostbridge -s 1:0,lpc \
    -s 2:0,virtio-net,tap1 -s 3:0,virtio-blk,./disk.img \
    -s 4:0,ahci-cd,./install.iso -c 4 -m 1024M \
    -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd,/path/to/vm-
image/BHYVE_UEFI_VARS.fd \
    guest
```

> Some Linux distributions require the use of UEFI variables to store the path for their UEFI boot file (using `linux64.efi` or `grubx64.efi` instead of `bootx64.efi`, for example). It is therefore recommended to use a variables file for Linux virtual machines to avoid having to manually alter the boot partition files.

To view or modify the variables file contents, use efivar(8) from the host.

sysutils/bhyve-firmware also contains a CSM-enabled firmware, to boot guests with no UEFI support in legacy BIOS mode:

```
# bhyve -AHP -s 0:0,hostbridge -s 1:0,lpc \
    -s 2:0,virtio-net,tap1 -s 3:0,virtio-blk,./disk.img \
    -s 4:0,ahci-cd,./install.iso -c 4 -m 1024M \
    -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI_CSM.fd \
    guest
```

25.7.5. Graphical UEFI Framebuffer for bhyve Guests

The UEFI firmware support is particularly useful with predominantly graphical guest operating systems such as Microsoft Windows®.

Support for the UEFI-GOP framebuffer may also be enabled with the `-s 29,fbuf,tcp=0.0.0.0:5900` flags. The framebuffer resolution may be configured with `w=800` and `h=600`, and bhyve can be instructed to wait for a VNC connection before booting the guest by adding `wait`. The framebuffer may be accessed from the host or over the network via the VNC protocol. Additionally, `-s 30,xhci,tablet` can be added to achieve precise mouse cursor synchronization with the host.

The resulting bhyve command would look like this:

```
# bhyve -AHP -s 0:0,hostbridge -s 31:0,lpc \
    -s 2:0,virtio-net,tap1 -s 3:0,virtio-blk,./disk.img \
    -s 4:0,ahci-cd,./install.iso -c 4 -m 1024M \
    -s 29,fbuf,tcp=0.0.0.0:5900,w=800,h=600,wait \
    -s 30,xhci,tablet \
    -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd \
    guest
```

Note, in BIOS emulation mode, the framebuffer will cease receiving updates once control is passed from firmware to guest operating system.

25.7.6. Creating a Microsoft Windows® Guest

Setting up a guest for Windows versions 10 or earlier can be done directly from the original installation media and is a relatively straightforward process. Aside from minimum resource requirements, running Windows as guest requires

- wiring virtual machine memory (flag `-w`) and
- booting with an UEFI bootrom.

An example for booting a virtual machine guest with a Windows installation ISO:

```
bhyve \
      -c 2 \
      -s 0,hostbridge \
      -s 3,nvme,windows2016.img \
      -s 4,ahci-cd,install.iso \
      -s 10,virtio-net,tap0 \
      -s 31,lpc \
      -s 30,xhci,tablet \
      -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd \
      -m 8G -H -w \
      windows2016
```

Only one or two VCPUs should be used during installation but this number can be increased once

Windows is installed.

VirtIO drivers must be installed to use the defined `virtio-net` network interface. An alternative is to switch to E1000 (Intel E82545) emulation by changing `virtio-net` to `e1000` in the above command line. However, performance will be impacted.

Creating a Windows 11 Guest

Beginning with Windows 11, Microsoft introduced a hardware requirement for a TPM 2 module. bhyve supports passing a hardware TPM through to a guest. The installation media can be modified to disable the relevant hardware checks. A detailed description for this process can be found on the FreeBSD Wiki.

Modifying Windows installation media and running Windows guests without a TPM module are unsupported by the manufacturer. Consider the application and use case before implementing such approaches.

25.7.7. Using ZFS with bhyve Guests

If ZFS is available on the host machine, using ZFS volumes instead of disk image files can provide significant performance benefits for the guest VMs. A ZFS volume can be created by:

```
# zfs create -V16G -o volmode=dev zroot/linuxdisk0
```

When starting the VM, specify the ZFS volume as the disk drive:

```
# bhyve -A -H -P -s 0:0,hostbridge -s 1:0,lpc -s 2:0,virtio-net,tap0 \
    -s3:0,virtio-blk,/dev/zvol/zroot/linuxdisk0 \
    -l com1,stdio -c 4 -m 1024M linuxguest
```

When using ZFS for the host as well as inside a guest, keep in mind the competing memory pressure of both systems caching the virtual machine's contents. To alleviate this, consider setting the host's ZFS filesystems to use metadata-only cache. To do this, apply the following settings to ZFS filesystems on the host, replacing `<name>` with the name of the specific zvol dataset name of the virtual machine.

```
# zfs set primarycache=metadata <name>
```

25.7.8. Creating a Virtual Machine Snapshot

Modern hypervisors allow their users to create "snapshots" of their state; such a snapshot includes a guest's disk, CPU, and memory contents. A snapshot can usually be taken independent of whether the guest is running or shut down. One can then reset and return the virtual machine to the precise state when the snapshot was taken.

ZFS Snapshots

Using ZFS volumes as the backing storage for a virtual machine enables the snapshotting of the guest's disk. For example:

```
zfs snapshot zroot/path/to/zvol@snapshot_name
```

Though it is possible to snapshot a ZFS volume this way while the guest is running, keep in mind that the contents of the virtual disk may be in an inconsistent state while the guest is active. It is therefore recommended to first shutdown or pause the guest before executing this command. Pausing a guest is not supported by default and needs to be enabled first (see Memory and CPU Snapshots)

> ⚠
>
> Rolling back a ZFS zvol to a snapshot while a virtual machine is using it may corrupt the file system contents and crash the guest. All unsaved data in the guest will be lost and modifications since the last snapshot may get destroyed.
>
> A second rollback may be required once the virtual machine is shut down to restore the file system to a useable state. This in turn will ultimately destroy any changes made after the snapshot.

Memory and CPU Snapshots (Experimental Feature)

As of FreeBSD 13, bhyve has an experimental "snapshot" feature for dumping a guest's memory and CPU state to a file and then halting the virtual machine. The guest can be resumed from the snapshot file contents later.

However, this feature is not enabled by default and requires the system to be rebuilt from source. See Building from Source for an in-depth description on the process of compiling the kernel with custom options.

> ⚠
>
> The functionality is not ready for production use and limited to work for specific virtual machine configurations. There are multiple limitations:
>
> - `nvme` and `virtio-blk` storage backends do not work yet
> - snapshots are only supported when the guest uses a single kind of each device, i.e. if there is more than one `ahci-hd` disk attached, snapshot creation will fail
> - additionally, the feature may be reasonably stable on Intel, but it probably won't work on AMD CPUs.

> ℹ
>
> Make sure the /usr/src directory is up-to date before taking the following steps. See Updating the Source for the detailed procedure how to do this.

First, add the following to /etc/src.conf:

```
WITH_BHYVE_SNAPHOT=yes
BHYVE_SNAPSHOT=1
```

```
MK_BHYVE_SNAPSHOT=yes
```

If the system was partially or wholly rebuilt, it is recommended to run

```
# cd /usr/src
# make cleanworld
```

before proceeding.

Then follow the steps outlined in the Quick Start section of the Updating FreeBSD from Source chapter to build and install world and kernel.

To verify successful activation of the snapshot feature, enter

```
# bhyvectl --usage
```

and check if the output lists a `--suspend` flag. If the flag is missing, the feature did not activate correctly.

Then, snapshot and suspend a running virtual machine of choice:

```
# bhyvectl --vm=vmname --suspend=/path/to/snapshot/filename
```

Provide an absolute path and filename to `--suspend`. Otherwise, bhyve will write snapshot data to whichever directory bhyve was started from.

Make sure to write the snapshot data to a secure directory. The generated output contains a full memory dump of the guest and may thus contain sensitive data (i.e. passwords)!

This creates three files:

- memory snapshot - named like the input to `--suspend`
- kernel file - name like the input to `--suspend` with the suffix .kern
- metadata - contains meta data about the system state, named with the suffix .meta

To restore a guest from a snapshot, use the `-r` flag with `bhyve`:

```
# bhyve -r /path/to/snapshot/filename
```

Restoring a guest snapshot on a different CPU architecture will not work. Generally, attempting to restore on a system not identical to the snapshot creator will likely fail.

25.7.9. Jailing bhyve

For improved security and separation of virtual machines from the host operating system, it is possible to run bhyve in a jail. See Jails for an in-depth description of jails and their security benefits.

Creating a Jail for bhyve

First, create a jail environment. If using a UFS file system, simply run:

```
# mkdir -p /jails/bhyve
```

If using a ZFS filesystem, use the following commands:

```
# zfs create zroot/jails
# zfs create zroot/jails/bhyve
```

Then create a ZFS zvol for the virtual machine `bhyvevm0`:

```
# zfs create zroot/vms
# zfs create -V 20G zroot/vms/bhyvevm0
```

If not using ZFS, use the following commands to create a disk image file directly in the jail directory structure:

```
# mkdir /jails/bhyve/vms
# truncate -s 20G /jails/bhyve/vms/bhyvevm0
```

Download a FreeBSD image, preferably a version equal to or older than the host and extract it into the jail directory:

```
# cd /jails
# fetch -o base.txz http://ftp.freebsd.org/pub/FreeBSD/releases/amd64/13.2-
RELEASE/base.txz
# tar -C /jails/bhyve -xvf base.txz
```

Running a higher FreeBSD version in a jail than the host is unsupported (i.e. running 14.0-RELEASE in a jail, embedded in a 13.2-RELEASE host).

Next, add a devfs ruleset to /etc/devfs.rules:

```
[devfsrules_jail_bhyve=100]
add include $devfsrules_hide_all
add include $devfsrules_unhide_login
add path 'urandom' unhide
add path 'random' unhide
add path 'crypto' unhide
add path 'shm' unhide
add path 'zero' unhide
add path 'null' unhide
add path 'mem' unhide
add path 'vmm' unhide
add path 'vmm/*' unhide
add path 'vmm.io' unhide
add path 'vmm.io/*' unhide
add path 'nmdmbhyve*' unhide
add path 'zvol' unhide
add path 'zvol/zroot' unhide
add path 'zvol/zroot/vms' unhide
add path 'zvol/zroot/vms/bhyvevm0' unhide
add path 'zvol/zroot/vms/bhyvevm1' unhide
add path 'tap10*' unhide
```

If there's another devfs rule with the numeric ID 100 in the /etc/devfs.rules file, replace the one in the listing with another yet unused ID number.

If not using a ZFS filesystem, skip the related zvol rules in /etc/devfs.rules:

```
add path 'zvol' unhide
add path 'zvol/zroot' unhide
add path 'zvol/zroot/vms' unhide
add path 'zvol/zroot/vms/bhyvevm0' unhide
add path 'zvol/zroot/vms/bhyvevm1' unhide
```

These rules will cause bhyve to

- create a virtual machine with disk volumes called `bhyvevm0` and `bhyvevm1`,
- use tap network interfaces with the name prefix `tap10`. That means, valid interface names will be `tap10`, `tap100`, `tap101`, … `tap109`, `tap1000` and so on.

  Limiting the access to a subset of possible tap interface names will prevent the jail (and thus bhyve) from seeing tap interfaces of the host and other jails.
- use nmdm devices prefixed with "bhyve", i.e. /dev/nmdmbhyve0.

Those rules can be expanded and varied with different guest and interface names as desired.

When intending to use bhyve on the host as well as in a one or more jails, remember that tap and nmdm interface names will operate in a shared environment. For example, use /dev/nmdmbhyve0 only either for bhyve on the host or in a jail.

Restart devfs for the changes to be loaded:

```
# service devfs restart
```

Then add a definition for the new jail into /etc/jail.conf or /etc/jail.conf.d. Replace the interface number $if and IP address with personal variations.

Example 33. Using NAT or routed traffic with a firewall

```
bhyve {
        $if = 0;
        exec.prestart = "/sbin/ifconfig epair${if} create up";
        exec.prestart += "/sbin/ifconfig epair${if}a up";
        exec.prestart += "/sbin/ifconfig epair${if}a name ${name}0";
        exec.prestart += "/sbin/ifconfig epair${if}b name jail${if}";
        exec.prestart += "/sbin/ifconfig ${name}0 inet 192.168.168.1/27";
        exec.prestart += "/sbin/sysctl net.inet.ip.forwarding=1";

        exec.clean;

        host.hostname = "the-hostname-here";
        vnet;
        vnet.interface = "jail${if}";
        path = "/jails/${name}";
        persist;
        securelevel = 3;
        devfs_ruleset = 100;
        mount.devfs;

        allow.vmm;

        exec.start += "/bin/sh /etc/rc";
        exec.stop = "/bin/sh /etc/rc.shutdown";

        exec.poststop += "/sbin/ifconfig ${name}0 destroy";
}
```

This example assumes use of a firewall like `pf` or `ipfw` to NAT the jail traffic. See the Firewalls chapter for more details on the available options to implement this.

Example 34. Using a bridged network connection

```
bhyve {
        $if = 0;
        exec.prestart = "/sbin/ifconfig epair${if} create up";
        exec.prestart += "/sbin/ifconfig epair${if}a up";
        exec.prestart += "/sbin/ifconfig epair${if}a name ${name}0";
        exec.prestart += "/sbin/ifconfig epair${if}b name jail${if}";
        exec.prestart += "/sbin/ifconfig bridge0 addm ${name}0";
        exec.prestart += "/sbin/sysctl net.inet.ip.forwarding=1";

        exec.clean;

        host.hostname = "the-hostname-here";
        vnet;
        vnet.interface = "jail${if}";
        path = "/jails/${name}";
        persist;
        securelevel = 3;
        devfs_ruleset = 100;
        mount.devfs;

        allow.vmm;

        exec.start += "/bin/sh /etc/rc";
        exec.stop = "/bin/sh /etc/rc.shutdown";

        exec.poststop += "/sbin/ifconfig ${name}0 destroy";
}
```

Having previously replaced the devfs ruleset ID 100 in /etc/devfs.rules with a custom unique number, remember to replace the numeric ID also in the jails.conf too.

Configuring the Jail

To start the jail for the first time and do some additional configuration work, enter:

```
# cp /etc/resolv.conf /jails/bhyve/etc
# service jail onestart bhyve
# jexec bhyve
# sysrc ifconfig_jail0="inet 192.168.168.2/27"
# sysrc defaultrouter="192.168.168.1"
# sysrc sendmail_enable=NONE
# sysrc cloned_interfaces="tap100"
# exit
```

Restart and enable the jail:

```
# sysrc jail_enable=YES
# service jail restart bhyve
```

Afterwards, create a virtual machine within the jail. For a FreeBSD guest, download an installation

ISO first:

```
# jexec bhyve
# cd /vms
# fetch -o freebsd.iso https://download.freebsd.org/releases/ISO-IMAGES/14.0/FreeBSD-
14.0-RELEASE-amd64-bootonly.iso
```

Creating a Virtual Machine Inside the Jail

To create a virtual machine, use `bhyvectl` to initialize it first:

```
# jexec bhyve
# bhyvectl --create --vm=bhyvevm0
```

> Creating the guest with `bhyvectl` may be required when initiating the virtual machine from a jail. Skipping this step may cause the following error message when starting `bhyve`:
>
> `vm_open: vm-name could not be opened. No such file or directory`

Finally, use the preferred way of starting the guest.

Example 35. Starting with `vmrun.sh` and ZFS

Using `vmrun.sh` on a ZFS filesystems:

```
# jexec bhyve
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M \
     -t tap100 -d /dev/zvol/zroot/vms/bhyvevm0 -i -I /vms/FreeBSD-14.0-RELEASE-
amd64-bootonly.iso bhyvevm0
```

Example 36. Starting with `vmrun.sh` and UFS

Using `vmrun.sh` on a UFS filesystem:

```
# jexec bhyve
# sh /usr/share/examples/bhyve/vmrun.sh -c 1 -m 1024M \
     -t tap100 -d /vms/bhyvevm0 -i -I /vms/FreeBSD-14.0-RELEASE-amd64-bootonly.iso
bhyvevm0
```

Example 37. Starting bhyve for an UEFI guest with ZFS

When wanting to use an UEFI guest, remember to first install the required firmware package sysutils/bhyve-firmware in the jail:

```
# pkg -j bhyve install bhyve-firmware
```

Then use `bhyve` directly:

```
# bhyve -A -c 4 -D -H -m 2G \
        -s 0,hostbridge \
        -s 1,lpc \
        -s 2,virtio-net,tap100 \
        -s 3,virtio-blk,/dev/zvol/zroot/vms/bhyvevm0 \
    -s 4,ahci-cd,/vms/FreeBSD-14.0-RELEASE-amd64-bootonly.iso \
        -s 31,fbuf,tcp=127.0.0.1:5900,w=1024,h=800,tablet \
        -l bootrom,/usr/local/share/uefi-firmware/BHYVE_UEFI.fd \
        -l com1,/dev/nmdbbhyve0A \
        bhyvevm0
```

This allows connecting to the virtual machine `bhyvevm0` through VNC as well as a serial console at /dev/nmdbbhyve0B.

25.7.10. Virtual Machine Consoles

It is advantageous to wrap the bhyve console in a session management tool such as sysutils/tmux or sysutils/screen in order to detach and reattach to the console. It is also possible to have the console of bhyve be a null modem device that can be accessed with `cu`. To do this, load the nmdm kernel module and replace `-l com1,stdio` with `-l com1,/dev/nmdm0A`. The /dev/nmdm devices are created automatically as needed, where each is a pair, corresponding to the two ends of the null modem cable (/dev/nmdm0A and /dev/nmdm0B). See nmdm(4) for more information.

```
# kldload nmdm
# bhyve -A -H -P -s 0:0,hostbridge -s 1:0,lpc -s 2:0,virtio-net,tap0 -s 3:0,virtio-blk,./linux.img \
    -l com1,/dev/nmdm0A -c 4 -m 1024M linuxguest
# cu -l /dev/nmdm0B
Connected

Ubuntu 13.10 handbook ttyS0

handbook login:
```

To disconnect from a console, enter a newline (i.e. press `RETURN`) followed by tilde (`~`), and finally dot (`.`). Keep in mind that only the connection is dropped while the login session remains active. Another user connecting to the same console could therefore make use of any active sessions without having to first authenticate. For security reasons, it's therefore recommended to logout before disconnecting.

The number in the nmdm device path must be unique for each virtual machine and must not be used by any other processes before bhyve starts. The number can be chosen arbitrarily and does not need to be taken from a consecutive sequence of numbers. The device node pair (i.e. /dev/nmdm0a and /dev/nmdm0b) are created dynamically when bhyve connects its console and destroyed when it shuts down. Keep this in mind when creating scripts to start the virtual machines: make sure that all virtual machines are assigned unique nmdm devices.

25.7.11. Managing Virtual Machines

A device node is created in /dev/vmm for each virtual machine. This allows the administrator to easily see a list of the running virtual machines:

```
# ls -al /dev/vmm
total 1
dr-xr-xr-x   2 root  wheel    512 Mar 17 12:19 ./
dr-xr-xr-x  14 root  wheel    512 Mar 17 06:38 ../
crw-------   1 root  wheel  0x1a2 Mar 17 12:20 guestname
crw-------   1 root  wheel  0x19f Mar 17 12:19 linuxguest
crw-------   1 root  wheel  0x1a1 Mar 17 12:19 otherguest
```

A specified virtual machine can be destroyed using `bhyvectl`:

```
# bhyvectl --destroy --vm=guestname
```

Destroying a virtual machine this way means killing it immediately. Any unsaved data will be lost, open files and filesystems may get corrupted. To gracefully shut down a virtual machine, send a `TERM` signal to its bhyve process instead. This triggers an ACPI shutdown event for the guest:

```
# ps ax | grep bhyve
17424  -  SC      56:48.27 bhyve: guestvm (bhyve)
# kill 17424
```

25.7.12. Tools and Utilities

There are numerous utilities and applications available in ports to help simplify setting up and managing bhyve virtual machines:

Table 34. bhyve Managers

Name	**License**	**Package**	**Documentation**
vm-bhyve	BSD-2	sysutils/vm-bhyve	Documentation
CBSD	BSD-2	sysutils/cbsd	Documentation
Virt-Manager	LGPL-3	deskutils/virt-manager	Documentation
Bhyve RC Script	Unknown	sysutils/bhyve-rc	Documentation
bmd	BSD-2	sysutils/bmd	Documentation
vmstated	BSD-2	sysutils/vmstated	Documentation

25.7.13. Persistent Configuration

In order to configure the system to start bhyve guests at boot time, some configuration file changes are required.

1. /etc/sysctl.conf

   When using tap interfaces as network backend, either manually set each used tap interface to UP or simply set the following sysctl:

   ```
   net.link.tap.up_on_open=1
   ```

2. /etc/rc.conf

   To connect the virtual machine's tap device to the network via a bridge, persisting the device settings in /etc/rc.conf is needed. Additionally, load the necessary kernel modules `vmm` for bhyve and `nmdm` for nmdm devices through the `kld_list` configuration variable. When configuring `ifconfig_bridge0`, make sure to replace `<ipaddr>/<netmask>` with the actual IP address of the physical interface (igb0 in this example) and remove IP settings from the physical device.

   ```
   # sysrc cloned_interfaces+="bridge0 tap0"
   # sysrc ifconfig_bridge0="inet <ipaddr>/<netmask> addm igb0 addm tap0"
   # sysrc kld_list+="nmdm vmm"
   # sysrc ifconfig_igb0="up"
   ```

Example 38. Setting the IP for a bridge device

For a host with an *igb0* interface connected to the network with IP `10.10.10.1` and netmask `255.255.255.0`, use the following commands:

```
# sysrc ifconfig_igb0="up"
# sysrc ifconfig_bridge0="inet 10.10.10.1/24 addm igb0 addm tap0"
# sysrc kld_list+="nmdm vmm"
# sysrc cloned_interfaces+="bridge0 tap0"
```

Modifying the IP address configuration of a system may terminate the current remote connection (e.g., via SSH), causing a lock out. Take precautions to maintain system access or make those modifications while logged in on a local terminal session.

25.8. FreeBSD as a Xen™-Host

Xen is a GPLv2-licensed type 1 hypervisor for Intel® and ARM® architectures. FreeBSD has included i386™ and AMD® 64-Bit DomU and Amazon EC2 unprivileged domain (virtual machine) support since FreeBSD 8.0 and includes Dom0 control domain (host) support in FreeBSD 11.0.

Support for para-virtualized (PV) domains has been removed from FreeBSD 11 in favor of hardware virtualized (HVM) domains, which provides better performance.

Xen™ is a bare-metal hypervisor, which means that it is the first program loaded after the BIOS. A special privileged guest called the Domain-0 (`Dom0` for short) is then started. The Dom0 uses its special privileges to directly access the underlying physical hardware, making it a high-performance solution. It is able to access the disk controllers and network adapters directly. The Xen™ management tools to manage and control the Xen™ hypervisor are also used by the Dom0 to create, list, and destroy VMs. Dom0 provides virtual disks and networking for unprivileged domains, often called `DomU`. Xen™ Dom0 can be compared to the service console of other hypervisor solutions, while the DomU is where individual guest VMs are run.

Xen™ can migrate VMs between different Xen™ servers. When the two xen hosts share the same underlying storage, the migration can be done without having to shut the VM down first. Instead, the migration is performed live while the DomU is running and there is no need to restart it or plan a downtime. This is useful in maintenance scenarios or upgrade windows to ensure that the services provided by the DomU are still provided. Many more features of Xen™ are listed on the Xen Wiki Overview page. Note that not all features are supported on FreeBSD yet.

25.8.1. Hardware Requirements for Xen™ Dom0

To run the Xen™ hypervisor on a host, certain hardware functionality is required. Running FreeBSD as a Xen host (Dom0) require Intel Extended Page Tables (EPT) or AMD Nested Page Tables (NPT) and Input/Output Memory Management Unit (IOMMU) support in the host processor.

In order to run a FreeBSD 13 Xen™ Dom0 the box must be booted using legacy boot (BIOS). FreeBSD 14 and newer supports booting as a Xen™ Dom0 in both BIOS and UEFI modes.

25.8.2. Xen™ Dom0 Control Domain Setup

Users should install the emulators/xen-kernel and sysutils/xen-tools packages, based on Xen™ 4.18.

Configuration files must be edited to prepare the host for the Dom0 integration after the Xen packages are installed. An entry to /etc/sysctl.conf disables the limit on how many pages of memory are allowed to be wired. Otherwise, DomU VMs with higher memory requirements will not run.

```
# echo 'vm.max_wired=-1' >> /etc/sysctl.conf
```

Another memory-related setting involves changing /etc/login.conf, setting the `memorylocked` option to `unlimited`. Otherwise, creating DomU domains may fail with `Cannot allocate memory` errors. After making the change to /etc/login.conf, run `cap_mkdb` to update the capability database. See Resource Limits for details.

```
# sed -i '' -e 's/memorylocked=64K/memorylocked=unlimited/' /etc/login.conf
# cap_mkdb /etc/login.conf
```

Add an entry for the Xen™ console to /etc/ttys:

```
# echo 'xc0     "/usr/libexec/getty Pc"         xterm   onifconsole  secure' >>
/etc/ttys
```

Selecting a Xen™ kernel in /boot/loader.conf activates the Dom0. Xen™ also requires resources like CPU and memory from the host machine for itself and other DomU domains. How much CPU and memory depends on the individual requirements and hardware capabilities. In this example, 8 GB of memory and 4 virtual CPUs are made available for the Dom0. The serial console is also activated, and logging options are defined.

The following command is used for Xen 4.7 packages:

```
# echo 'hw.pci.mcfg=0' >> /boot/loader.conf
# echo 'if_tap_load="YES"' >> /boot/loader.conf
# echo 'xen_kernel="/boot/xen"' >> /boot/loader.conf
# echo 'xen_cmdline="dom0_mem=8192M dom0_max_vcpus=4 dom0pvh=1 console=com1,vga
com1=115200,8n1 guest_loglvl=all loglvl=all"' >> /boot/loader.conf
```

For Xen versions 4.11 and higher, the following command should be used instead:

```
# echo 'if_tap_load="YES"' >> /boot/loader.conf
# echo 'xen_kernel="/boot/xen"' >> /boot/loader.conf
# echo 'xen_cmdline="dom0_mem=8192M dom0_max_vcpus=4 dom0=pvh console=com1,vga
com1=115200,8n1 guest_loglvl=all loglvl=all"' >> /boot/loader.conf
```

Log files that Xen™ creates for the DomU VMs are stored in /var/log/xen. Please be sure to check the contents of that directory if experiencing issues.

Activate the xencommons service during system startup:

```
# sysrc xencommons_enable=yes
```

These settings are enough to start a Dom0-enabled system. However, it lacks network functionality for the DomU machines. To fix that, define a bridged interface with the main NIC of the system which the DomU VMs can use to connect to the network. Replace *em0* with the host network interface name.

```
# sysrc cloned_interfaces="bridge0"
# sysrc ifconfig_bridge0="addm em0 SYNCDHCP"
# sysrc ifconfig_em0="up"
```

Restart the host to load the Xen™ kernel and start the Dom0.

```
# shutdown -r now
```

After successfully booting the Xen™ kernel and logging into the system again, the Xen™ management tool `xl` is used to show information about the domains.

```
# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  8192     4     r-----     962.0
```

The output confirms that the Dom0 (called `Domain-0`) has the ID `0` and is running. It also has the memory and virtual CPUs that were defined in /boot/loader.conf earlier. More information can be found in the Xen™ Documentation. DomU guest VMs can now be created.

25.8.3. Xen™ DomU Guest VM Configuration

Unprivileged domains consist of a configuration file and virtual or physical hard disks. Virtual disk storage for the DomU can be files created by truncate(1) or ZFS volumes as described in “Creating and Destroying Volumes”. In this example, a 20 GB volume is used. A VM is created with the ZFS volume, a FreeBSD ISO image, 1 GB of RAM and two virtual CPUs. The ISO installation file is retrieved with fetch(1) and saved locally in a file called freebsd.iso.

```
# fetch https://download.freebsd.org/releases/ISO-IMAGES/14.0/FreeBSD-14.0-RELEASE-
amd64-bootonly.iso -o freebsd.iso
```

A ZFS volume of 20 GB called xendisk0 is created to serve as the disk space for the VM.

```
# zfs create -V20G -o volmode=dev zroot/xendisk0
```

The new DomU guest VM is defined in a file. Some specific definitions like name, keymap, and VNC connection details are also defined. The following freebsd.cfg contains a minimum DomU configuration for this example:

```
# cat freebsd.cfg
builder = "hvm" ①
name = "freebsd" ②
memory = 1024 ③
vcpus = 2 ④
vif = [ 'mac=00:16:3E:74:34:32,bridge=bridge0' ] ⑤
disk = [
'/dev/zvol/tank/xendisk0,raw,hda,rw', ⑥
'/root/freebsd.iso,raw,hdc:cdrom,r' ⑦
  ]
vnc = 1 ⑧
vnclisten = "0.0.0.0"
serial = "pty"
```

```
usbdevice = "tablet"
```

These lines are explained in more detail:

① This defines what kind of virtualization to use. `hvm` refers to hardware-assisted virtualization or hardware virtual machine. Guest operating systems can run unmodified on CPUs with virtualization extensions, providing nearly the same performance as running on physical hardware. `generic` is the default value and creates a PV domain.

② Name of this virtual machine to distinguish it from others running on the same Dom0. Required.

③ Quantity of RAM in megabytes to make available to the VM. This amount is subtracted from the hypervisor's total available memory, not the memory of the Dom0.

④ Number of virtual CPUs available to the guest VM. For best performance, do not create guests with more virtual CPUs than the number of physical CPUs on the host.

⑤ Virtual network adapter. This is the bridge connected to the network interface of the host. The `mac` parameter is the MAC address set on the virtual network interface. This parameter is optional, if no MAC is provided Xen™ will generate a random one.

⑥ Full path to the disk, file, or ZFS volume of the disk storage for this VM. Options and multiple disk definitions are separated by commas.

⑦ Defines the Boot medium from which the initial operating system is installed. In this example, it is the ISO image downloaded earlier. Consult the Xen™ documentation for other kinds of devices and options to set.

⑧ Options controlling VNC connectivity to the serial console of the DomU. In order, these are: active VNC support, define IP address on which to listen, device node for the serial console, and the input method for precise positioning of the mouse and other input methods. `keymap` defines which keymap to use, and is `english` by default.

After the file has been created with all the necessary options, the DomU is created by passing it to `xl create` as a parameter.

```
# xl create freebsd.cfg
```

Each time the Dom0 is restarted, the configuration file must be passed to `xl create` again to re-create the DomU. By default, only the Dom0 is created after a reboot, not the individual VMs. The VMs can continue where they left off as they stored the operating system on the virtual disk. The virtual machine configuration can change over time (for example, when adding more memory). The virtual machine configuration files must be properly backed up and kept available to be able to re-create the guest VM when needed.

The output of `xl list` confirms that the DomU has been created.

```
# xl list
Name                                        ID   Mem VCPUs      State   Time(s)
Domain-0                                     0  8192     4     r-----    1653.4
```

```
freebsd                                          1  1024     1     -b----    663.9
```

To begin the installation of the base operating system, start the VNC client, directing it to the main network address of the host or to the IP address defined on the `vnclisten` line of freebsd.cfg. After the operating system has been installed, shut down the DomU and disconnect the VNC viewer. Edit freebsd.cfg, removing the line with the `cdrom` definition or commenting it out by inserting a `#` character at the beginning of the line. To load this new configuration, it is necessary to remove the old DomU with `xl destroy`, passing either the name or the id as the parameter. Afterwards, recreate it using the modified **freebsd.cfg**.

```
# xl destroy freebsd
# xl create freebsd.cfg
```

The machine can then be accessed again using the VNC viewer. This time, it will boot from the virtual disk where the operating system has been installed and can be used as a virtual machine.

25.8.4. Troubleshooting

This section contains basic information in order to help troubleshoot issues found when using FreeBSD as a Xen™ host or guest.

Host Boot Troubleshooting

Please note that the following troubleshooting tips are intended for Xen™ 4.11 or newer. When still using Xen™ 4.7 and having issues, consider migrating to a newer version of Xen™.

In order to troubleshoot host boot issues, a serial cable or a debug USB cable is needed. Verbose Xen™ boot output can be obtained by adding options to the `xen_cmdline` option found in loader.conf. A couple of relevant debug options are:

- `iommu=debug`: can be used to print additional diagnostic information about the iommu.
- `dom0=verbose`: can be used to print additional diagnostic information about the dom0 build process.
- `sync_console`: flag to force synchronous console output. Useful for debugging to avoid losing messages due to rate limiting. Never use this option in production environments since it can allow malicious guests to perform DoS attacks against Xen™ using the console.

FreeBSD should also be booted in verbose mode in order to identify any issues. To activate verbose booting, run this command:

```
# echo 'boot_verbose="YES"' >> /boot/loader.conf
```

If none of these options help solving the problem, please send the serial boot log to freebsd-xen@FreeBSD.org and xen-devel@lists.xenproject.org for further analysis.

Guest Creation Troubleshooting

Issues can also arise when creating guests, the following attempts to provide some help for those trying to diagnose guest creation issues.

The most common cause of guest creation failures is the `xl` command spitting some error and exiting with a return code different than 0. If the error provided is not enough to help identify the issue, more verbose output can also be obtained from `xl` by using the `v` option repeatedly.

```
# xl -vvv create freebsd.cfg
Parsing config from freebsd.cfg
libxl: debug: libxl_create.c:1693:do_domain_create: Domain 0:ao 0x800d750a0: create:
how=0x0 callback=0x0 poller=0x800d6f0f0
libxl: debug: libxl_device.c:397:libxl__device_disk_set_backend: Disk vdev=xvda
spec.backend=unknown
libxl: debug: libxl_device.c:432:libxl__device_disk_set_backend: Disk vdev=xvda, using
backend phy
libxl: debug: libxl_create.c:1018:initiate_domain_create: Domain 1:running bootloader
libxl: debug: libxl_bootloader.c:328:libxl__bootloader_run: Domain 1:not a PV/PVH
domain, skipping bootloader
libxl: debug: libxl_event.c:689:libxl__ev_xswatch_deregister: watch w=0x800d96b98:
deregister unregistered
domainbuilder: detail: xc_dom_allocate: cmdline="", features=""
domainbuilder: detail: xc_dom_kernel_file: filename
="/usr/local/lib/xen/boot/hvmloader"
domainbuilder: detail: xc_dom_malloc_filemap    : 326 kB
libxl: debug: libxl_dom.c:988:libxl__load_hvm_firmware_module: Loading BIOS:
/usr/local/share/seabios/bios.bin
...
```

If the verbose output does not help diagnose the issue, there are also QEMU and Xen™ toolstack logs in /var/log/xen. Note that the name of the domain is appended to the log name, so if the domain is named `freebsd` find a /var/log/xen/xl-freebsd.log and likely a /var/log/xen/qemu-dm-freebsd.log. Both log files can contain useful information for debugging. If none of this helps solve the issue, please send the description of the issue and as much information as possible to freebsd-xen@FreeBSD.org and xen-devel@lists.xenproject.org to get help.

Chapter 26. Localization - i18n/L10n Usage and Setup

26.1. Synopsis

FreeBSD is a distributed project with users and contributors located all over the world. As such, FreeBSD supports localization into many languages, allowing users to view, input, or process data in non-English languages. One can choose from most of the major languages, including, but not limited to: Chinese, German, Japanese, Korean, French, Russian, and Vietnamese.

The term internationalization has been shortened to i18n, which represents the number of letters between the first and the last letters of `internationalization`. L10n uses the same naming scheme, but from `localization`. The i18n/L10n methods, protocols, and applications allow users to use languages of their choice.

This chapter discusses the internationalization and localization features of FreeBSD.

Read this chapter to learn:

- How locale names are constructed.
- How to set the locale for a login shell.
- How to configure the console for non-English languages.
- How to configure Xorg for different languages.
- How to find i18n-compliant applications.
- Where to find more information for configuring specific languages.

Before reading this chapter:

- Know how to install additional third-party applications.

26.2. Using Localization

Localization settings are based on three components: the language code, country code, and encoding. Locale names are constructed from these parts as follows:

```
LanguageCode_CountryCode.Encoding
```

The *LanguageCode* and *CountryCode* are used to determine the country and the specific language variation. Common Language and Country Codes provides some examples of *LanguageCode_CountryCode*:

Table 35. Common Language and Country Codes

LanguageCode_Country Code	Description
en_US	English, United States
ru_RU	Russian, Russia
zh_TW	Traditional Chinese, Taiwan

A complete listing of available locales can be found by typing:

```
% locale -a | more
```

To determine the current locale setting:

```
% locale
```

Language specific character sets, such as ISO8859-1, ISO8859-15, KOI8-R, and CP437, are described in multibyte(3). The active list of character sets can be found at the IANA Registry.

Some languages, such as Chinese or Japanese, cannot be represented using ASCII characters and require an extended language encoding using either wide or multibyte characters. Examples of wide or multibyte encodings include EUC and Big5. Older applications may mistake these encodings for control characters while newer applications usually recognize these characters. Depending on the implementation, users may be required to compile an application with wide or multibyte character support, or to configure it correctly.

FreeBSD uses Xorg-compatible locale encodings.

The rest of this section describes the various methods for configuring the locale on a FreeBSD system. The next section will discuss the considerations for finding and compiling applications with i18n support.

26.2.1. Setting Locale for Login Shell

Locale settings are configured either in a user's ~/.login_conf or in the startup file of the user's shell: ~/.profile, ~/.bashrc, or ~/.cshrc.

Two environment variables should be set:

- `LANG`, which sets the locale
- `MM_CHARSET`, which sets the MIME character set used by applications

In addition to the user's shell configuration, these variables should also be set for specific application configuration and Xorg configuration.

Two methods are available for making the needed variable assignments: the login class method, which is the recommended method, and the startup file method. The next two sections demonstrate how to use both methods.

Login Classes Method

This first method is the recommended method as it assigns the required environment variables for locale name and MIME character sets for every possible shell. This setup can either be performed by each user or it can be configured for all users by the superuser.

This minimal example sets both variables for Latin-1 encoding in the .login_conf of an individual user's home directory:

```
me:\
    :charset=ISO-8859-1:\
    :lang=de_DE.ISO8859-1:
```

Here is an example of a user's ~/.login_conf that sets the variables for Traditional Chinese in BIG-5 encoding. More variables are needed because some applications do not correctly respect locale variables for Chinese, Japanese, and Korean:

```
#Users who do not wish to use monetary units or time formats
#of Taiwan can manually change each variable
me:\
    :lang=zh_TW.Big5:\

:setenv=LC_ALL=zh_TW.Big5,LC_COLLATE=zh_TW.Big5,LC_CTYPE=zh_TW.Big5,LC_MESSAGES=zh_TW.
Big5,LC_MONETARY=zh_TW.Big5,LC_NUMERIC=zh_TW.Big5,LC_TIME=zh_TW.Big5:\
    :charset=big5:\
    :xmodifiers="@im=gcin": #Set gcin as the XIM Input Server
```

Alternately, the superuser can configure all users of the system for localization. The following variables in /etc/login.conf are used to set the locale and MIME character set:

```
language_name|Account Type Description:\
    :charset=MIME_charset:\
    :lang=locale_name:\
    :tc=default:
```

So, the previous Latin-1 example would look like this:

```
german|German Users Accounts:\
    :charset=ISO-8859-1:\
    :lang=de_DE.ISO8859-1:\
    :tc=default:
```

See login.conf(5) for more details about these variables. Note that it already contains pre-defined *russian* class.

Whenever /etc/login.conf is edited, remember to execute the following command to update the capability database:

```
# cap_mkdb /etc/login.conf
```

For an end user, the `cap_mkdb` command will need to be run on their ~/.login_conf for any changes to take effect.

Utilities Which Change Login Classes

In addition to manually editing /etc/login.conf, several utilities are available for setting the locale for newly created users.

When using `vipw` to add new users, specify the *language* to set the locale:

```
user:password:1111:11:language:0:0:User Name:/home/user:/bin/sh
```

When using `adduser` to add new users, the default language can be pre-configured for all new users or specified for an individual user.

If all new users use the same language, set `defaultclass=language` in /etc/adduser.conf.

To override this setting when creating a user, either input the required locale at this prompt:

```
Enter login class: default []:
```

or specify the locale to set when invoking `adduser`:

```
# adduser -class language
```

If `pw` is used to add new users, specify the locale as follows:

```
# pw useradd user_name -L language
```

To change the login class of an existing user, `chpass` can be used. Invoke it as superuser and provide the username to edit as the argument.

```
# chpass user_name
```

Shell Startup File Method

This second method is not recommended as each shell that is used requires manual configuration, where each shell has a different configuration file and differing syntax. As an example, to set the German language for the `sh` shell, these lines could be added to ~/.profile to set the shell for that user only. These lines could also be added to /etc/profile or /usr/share/skel/dot.profile to set that shell for all users:

```
LANG=de_DE.ISO8859-1; export LANG
MM_CHARSET=ISO-8859-1; export MM_CHARSET
```

However, the name of the configuration file and the syntax used differs for the `csh` shell. These are the equivalent settings for ~/.login, /etc/csh.login, or /usr/share/skel/dot.login:

```
setenv LANG de_DE.ISO8859-1
setenv MM_CHARSET ISO-8859-1
```

To complicate matters, the syntax needed to configure Xorg in ~/.xinitrc also depends upon the shell. The first example is for the `sh` shell and the second is for the `csh` shell:

```
LANG=de_DE.ISO8859-1; export LANG
```

```
setenv LANG de_DE.ISO8859-1
```

26.2.2. Console Setup

Several localized fonts are available for the console. To see a listing of available fonts, type `ls /usr/share/syscons/fonts`. To configure the console font, specify the *font_name*, without the .fnt suffix, in /etc/rc.conf:

```
font8x16=font_name
font8x14=font_name
font8x8=font_name
```

The keymap and screenmap can be set by adding the following to /etc/rc.conf:

```
scrnmap=screenmap_name
keymap=keymap_name
keychange="fkey_number sequence"
```

To see the list of available screenmaps, type `ls /usr/share/syscons/scrnmaps`. Do not include the .scm suffix when specifying *screenmap_name*. A screenmap with a corresponding mapped font is usually needed as a workaround for expanding bit 8 to bit 9 on a VGA adapter's font character matrix so that letters are moved out of the pseudographics area if the screen font uses a bit 8 column.

To see the list of available keymaps, type `ls /usr/share/syscons/keymaps`. When specifying the *keymap_name*, do not include the .kbd suffix. To test keymaps without rebooting, use kbdmap(1).

The `keychange` entry is usually needed to program function keys to match the selected terminal type because function key sequences cannot be defined in the keymap.

Next, set the correct console terminal type in /etc/ttys for all virtual terminal entries. Defined Terminal Types for Character Sets summarizes the available terminal types.:

Table 36. Defined Terminal Types for Character Sets

Character Set	Terminal Type
ISO8859-1 or ISO8859-15	`cons25l1`
ISO8859-2	`cons25l2`
ISO8859-7	`cons25l7`
KOI8-R	`cons25r`
KOI8-U	`cons25u`
CP437 (VGA default)	`cons25`
US-ASCII	`cons25w`

For languages with wide or multibyte characters, install a console for that language from the FreeBSD Ports Collection. The available ports are summarized in Available Console from Ports Collection. Once installed, refer to the port's pkg-message or man pages for configuration and usage instructions.

Table 37. Available Console from Ports Collection

Language	Port Location
Traditional Chinese (BIG-5)	chinese/big5con
Chinese/Japanese/Korean	chinese/cce
Chinese/Japanese/Korean	chinese/zhcon
Japanese	chinese/kon2
Japanese	japanese/kon2-14dot
Japanese	japanese/kon2-16dot

If moused is enabled in /etc/rc.conf, additional configuration may be required. By default, the mouse cursor of the syscons(4) driver occupies the `0xd0`-`0xd3` range in the character set. If the language uses this range, move the cursor's range by adding the following line to /etc/rc.conf:

```
mousechar_start=3
```

26.2.3. Xorg Setup

The X Window System describes how to install and configure Xorg. When configuring Xorg for localization, additional fonts and input methods are available from the FreeBSD Ports Collection. Application specific i18n settings such as fonts and menus can be tuned in ~/.Xresources and should allow users to view their selected language in graphical application menus.

The X Input Method (XIM) protocol is an Xorg standard for inputting non-English characters. Available Input Methods summarizes the input method applications which are available in the FreeBSD Ports Collection. Additional Fcitx and Uim applications are also available.

Table 38. Available Input Methods

Language	Input Method
Chinese	chinese/gcin
Chinese	chinese/ibus-chewing
Chinese	chinese/ibus-pinyin
Chinese	chinese/oxim
Chinese	chinese/scim-fcitx
Chinese	chinese/scim-pinyin
Chinese	chinese/scim-tables
Japanese	japanese/ibus-anthy
Japanese	japanese/ibus-mozc
Japanese	japanese/ibus-skk
Japanese	japanese/im-ja
Japanese	japanese/kinput2
Japanese	japanese/scim-anthy
Japanese	japanese/scim-canna
Japanese	japanese/scim-honoka
Japanese	japanese/scim-honoka-plugin-romkan
Japanese	japanese/scim-honoka-plugin-wnn
Japanese	japanese/scim-prime
Japanese	japanese/scim-skk
Japanese	japanese/scim-tables
Japanese	japanese/scim-tomoe
Japanese	japanese/scim-uim
Japanese	japanese/skkinput
Japanese	japanese/skkinput3
Japanese	japanese/uim-anthy
Korean	korean/ibus-hangul
Korean	korean/imhangul
Korean	korean/nabi
Korean	korean/scim-hangul
Korean	korean/scim-tables

Language	Input Method
Vietnamese	vietnamese/xvnkb
Vietnamese	vietnamese/x-unikey

26.3. Finding i18n Applications

i18n applications are programmed using i18n kits under libraries. These allow developers to write a simple file and translate displayed menus and texts to each language.

The FreeBSD Ports Collection contains many applications with built-in support for wide or multibyte characters for several languages. Such applications include `i18n` in their names for easy identification. However, they do not always support the language needed.

Some applications can be compiled with the specific charset. This is usually done in the port's Makefile or by passing a value to configure. Refer to the i18n documentation in the respective FreeBSD port's source for more information on how to determine the needed configure value or the port's Makefile to determine which compile options to use when building the port.

26.4. Locale Configuration for Specific Languages

This section provides configuration examples for localizing a FreeBSD system for the Russian language. It then provides some additional resources for localizing other languages.

26.4.1. Russian Language (KOI8-R Encoding)

This section shows the specific settings needed to localize a FreeBSD system for the Russian language. Refer to Using Localization for a complete description of each type of setting.

To set this locale for the login shell, add the following lines to each user's ~/.login_conf:

```
me:My Account:\
    :charset=KOI8-R:\
    :lang=ru_RU.KOI8-R:
```

To configure the console, add the following lines to /etc/rc.conf:

```
keymap="ru.utf-8"
scrnmap="utf-82cp866"
font8x16="cp866b-8x16"
font8x14="cp866-8x14"
font8x8="cp866-8x8"
mousechar_start=3
```

For each `ttyv` entry in /etc/ttys, use `cons25r` as the terminal type.

To configure printing, a special output filter is needed to convert from KOI8-R to CP866 since most printers with Russian characters come with hardware code page CP866. FreeBSD includes a default filter for this purpose, /usr/libexec/lpr/ru/koi2alt. To use this filter, add this entry to /etc/printcap:

```
lp|Russian local line printer:\
    :sh:of=/usr/libexec/lpr/ru/koi2alt:\
    :lp=/dev/lpt0:sd=/var/spool/output/lpd:lf=/var/log/lpd-errs:
```

Refer to printcap(5) for a more detailed explanation.

To configure support for Russian filenames in mounted MS-DOS® file systems, include `-L` and the locale name when adding an entry to /etc/fstab:

```
/dev/ad0s2      /dos/c  msdos   rw,-Lru_RU.KOI8-R 0 0
```

Refer to mount_msdosfs(8) for more details.

To configure Russian fonts for Xorg, install the x11-fonts/xorg-fonts-cyrillic package. Then, check the `"Files"` section in /etc/X11/xorg.conf. The following line must be added *before* any other `FontPath` entries:

```
FontPath   "/usr/local/lib/X11/fonts/cyrillic"
```

Additional Cyrillic fonts are available in the Ports Collection.

To activate a Russian keyboard, add the following to the `"Keyboard"` section of /etc/xorg.conf:

```
Option "XkbLayout"   "us,ru"
Option "XkbOptions"  "grp:toggle"
```

Make sure that `XkbDisable` is commented out in that file.

For `grp:toggle` use `Right Alt`, for `grp:ctrl_shift_toggle` use `Ctrl`+`Shift`. For `grp:caps_toggle` use `CapsLock`. The old `CapsLock` function is still available in LAT mode only using `Shift`+`CapsLock`. `grp:caps_toggle` does not work in Xorg for some unknown reason.

If the keyboard has "Windows®" keys, and some non-alphabetical keys are mapped incorrectly, add the following line to /etc/xorg.conf:

```
Option "XkbVariant" ",winkeys"
```

The Russian XKB keyboard may not work with non-localized applications. Minimally localized applications should call a `XtSetLanguageProc (NULL, NULL, NULL);` function early in the program.

See http://koi8.pp.ru/xwin.html for more instructions on localizing Xorg applications. For more general information about KOI8-R encoding, refer to http://koi8.pp.ru/.

26.4.2. Additional Language-Specific Resources

This section lists some additional resources for configuring other locales.

Traditional Chinese for Taiwan

The FreeBSD-Taiwan Project has a Chinese HOWTO for FreeBSD at http://netlab.cse.yzu.edu.tw/~statue/freebsd/zh-tut/.

Greek Language Localization

A complete article on Greek support in FreeBSD is available here, in Greek only, as part of the official FreeBSD Greek documentation.

Japanese and Korean Language Localization

For Japanese, refer to http://www.jp.FreeBSD.org/, and for Korean, refer to http://www.kr.FreeBSD.org/.

Non-English FreeBSD Documentation

Some FreeBSD contributors have translated parts of the FreeBSD documentation to other languages. They are available through links on the FreeBSD web site or in /usr/share/doc.

Chapter 27. Updating and Upgrading FreeBSD

27.1. Synopsis

FreeBSD is under constant development between releases. Some people prefer to use the officially released versions, while others prefer to keep in sync with the latest developments. However, even official releases are often updated with security and other critical fixes. Regardless of the version used, FreeBSD provides all the necessary tools to keep the system updated, and allows for easy upgrades between versions. This chapter describes how to track the development system and the basic tools for keeping a FreeBSD system up-to-date.

Read this chapter to learn:

- How to keep a FreeBSD system up-to-date with freebsd-update or Git.
- How to compare the state of an installed system against a known pristine copy.
- How to keep the installed documentation up-to-date with Git or documentation ports.
- The difference between the two development branches: FreeBSD-STABLE and FreeBSD-CURRENT.
- How to rebuild and reinstall the entire base system.

Before reading this chapter:

- Properly set up the network connection (Advanced Networking).
- Know how to install additional third-party software (Installing Applications: Packages and Ports).

> Throughout this chapter, `git` is used to obtain and update FreeBSD sources. Optionally, the devel/git port or package may be used.

27.2. FreeBSD Update

Applying security patches in a timely manner and upgrading to a newer release of an operating system are important aspects of ongoing system administration. FreeBSD includes a utility called `freebsd-update` which can be used to perform both these tasks.

This utility supports binary security and errata updates to FreeBSD, without the need to manually compile and install the patch or a new kernel. Binary updates are available for all architectures and releases currently supported by the security team. The list of supported releases and their estimated end-of-life dates are listed at https://www.FreeBSD.org/security/.

This utility also supports operating system upgrades to minor point releases as well as upgrades to another release branch. Before upgrading to a new release, review its release announcement as it contains important information pertinent to the release. Release announcements are available from https://www.FreeBSD.org/releases/.

If a crontab(5) utilizing the features of freebsd-update(8) exists, it must be disabled before upgrading the operating system.

This section describes the configuration file used by `freebsd-update`, demonstrates how to apply a security patch and how to upgrade to a minor or major operating system release, and discusses some of the considerations when upgrading the operating system.

27.2.1. The Configuration File

The default configuration file for `freebsd-update` works as-is. Some users may wish to tweak the default configuration in /etc/freebsd-update.conf, allowing better control of the process. The comments in this file explain the available options, but the following may require a bit more explanation:

```
# Components of the base system which should be kept updated.
Components world kernel
```

This parameter controls which parts of FreeBSD will be kept up-to-date. The default is to update the entire base system and the kernel. Individual components can instead be specified, such as `src/base` or `src/sys`. However, the best option is to leave this at the default as changing it to include specific items requires every needed item to be listed. Over time, this could have disastrous consequences as source code and binaries may become out of sync.

```
# Paths which start with anything matching an entry in an IgnorePaths
# statement will be ignored.
IgnorePaths /boot/kernel/linker.hints
```

To leave specified directories, such as /bin or /sbin, untouched during the update process, add their paths to this statement. This option may be used to prevent `freebsd-update` from overwriting local modifications.

```
# Paths which start with anything matching an entry in an UpdateIfUnmodified
# statement will only be updated if the contents of the file have not been
# modified by the user (unless changes are merged; see below).
UpdateIfUnmodified /etc/ /var/ /root/ /.cshrc /.profile
```

This option will only update unmodified configuration files in the specified directories. Any changes made by the user will prevent the automatic updating of these files. There is another option, `KeepModifiedMetadata`, which will instruct `freebsd-update` to save the changes during the merge.

```
# When upgrading to a new FreeBSD release, files which match MergeChanges
# will have any local changes merged into the version from the new release.
MergeChanges /etc/ /var/named/etc/ /boot/device.hints
```

List of directories with configuration files that `freebsd-update` should attempt to merge. The file merge process is a series of diff(1) patches. Merges are either accepted, open an editor, or cause `freebsd-update` to abort. When in doubt, backup /etc and just accept the merges.

```
# Directory in which to store downloaded updates and temporary
# files used by FreeBSD Update.
# WorkDir /var/db/freebsd-update
```

This directory is where all patches and temporary files are placed. In cases where the user is doing a version upgrade, this location should have at least a gigabyte of disk space available.

```
# When upgrading between releases, should the list of Components be
# read strictly (StrictComponents yes) or merely as a list of components
# which *might* be installed of which FreeBSD Update should figure out
# which actually are installed and upgrade those (StrictComponents no)?
# StrictComponents no
```

When this option is set to `yes`, `freebsd-update` will assume that the `Components` list is complete and will not attempt to make changes outside of the list. Effectively, `freebsd-update` will attempt to update every file which belongs to the `Components` list.

Refer to freebsd-update.conf(5) for more details.

27.2.2. Applying Security Patches

The process of applying FreeBSD security patches has been simplified, allowing an administrator to keep a system fully patched using `freebsd-update`. More information about FreeBSD security advisories can be found in FreeBSD Security Advisories.

FreeBSD security patches may be downloaded and installed using the following commands. The first command will determine if any outstanding patches are available, and if so, will list the files that will be modified if the patches are applied. The second command will apply the patches.

```
# freebsd-update fetch
# freebsd-update install
```

If the update applies any kernel patches, the system will need a reboot in order to boot into the patched kernel. If the patch was applied to any running binaries, the affected applications should be restarted so that the patched version of the binary is used.

> Usually, the user needs to be prepared to reboot the system. To know if the system requires a reboot due to a kernel update, execute the commands `freebsd-version -k` and `uname -r`. Reboot the system if the outputs differ.

The system can be configured to automatically check for updates once every day by adding this entry to /etc/crontab:

```
@daily                                  root    freebsd-update cron
```

If patches exist, they will automatically be downloaded but will not be applied. The `root` user will be sent an email so that the patches may be reviewed and manually installed with `freebsd-update install`.

If anything goes wrong, `freebsd-update` has the ability to roll back the last set of changes with the following command:

```
# freebsd-update rollback
Uninstalling updates... done.
```

Again, the system should be restarted if the kernel or any kernel modules were modified and any affected binaries should be restarted.

Only the GENERIC kernel can be automatically updated by `freebsd-update`. If a custom kernel is installed, it will have to be rebuilt and reinstalled after `freebsd-update` finishes installing the updates. The default kernel name is *GENERIC*. The uname(1) command may be used to verify its installation.

> ℹ Always keep a copy of the GENERIC kernel in /boot/GENERIC. It will be helpful in diagnosing a variety of problems and in performing version upgrades. Refer to Custom Kernels with FreeBSD 9.X and Later for instructions on how to get a copy of the GENERIC kernel.

Unless the default configuration in /etc/freebsd-update.conf has been changed, `freebsd-update` will install the updated kernel sources along with the rest of the updates. Rebuilding and reinstalling a new custom kernel can then be performed in the usual way.

The updates distributed by `freebsd-update` do not always involve the kernel. It is not necessary to rebuild a custom kernel if the kernel sources have not been modified by `freebsd-update install`. However, `freebsd-update` will always update /usr/src/sys/conf/newvers.sh. The current patch level, as indicated by the `-p` number reported by `uname -r`, is obtained from this file. Rebuilding a custom kernel, even if nothing else changed, allows `uname` to accurately report the current patch level of the system. This is particularly helpful when maintaining multiple systems, as it allows for a quick assessment of the updates installed in each one.

27.2.3. Performing Minor and Major Version Upgrades

Upgrades from one minor version of FreeBSD to another are called *minor version* upgrades. An example:

- FreeBSD 13.1 to 13.2.

Major version upgrades increase the major version number. An example:

- FreeBSD 13.2 to 14.0.

Both types of upgrade can be performed by providing `freebsd-update` with a release version target.

After each new `RELEASE`, FreeBSD package build servers will, for a limited period, **not** use the newer version of the operating system. This provides continuity for the many users who do not upgrade immediately after a release announcement. For example:

- packages for users of 13.1 and 13.2 will be built on a server running 13.1, until 13.1 reaches end of life

— and, critically:

- a kernel module that is built on 13.1 might **not** be suitable for 13.2.

So, with any minor or major OS upgrade, if the package requirements include any kernel module:

- **be prepared to build the module from source**.

If the system is running a custom kernel, make sure that a copy of the GENERIC kernel exists in /boot/GENERIC before starting the upgrade. Refer to Custom Kernels with FreeBSD 9.X and Later for instructions on how to get a copy of the GENERIC kernel.

Before upgrading to a new version, ensure the existing FreeBSD installation is up to date with respect to security and errata patches:

```
# freebsd-update fetch
# freebsd-update install
```

The following command, when run on a FreeBSD 13.1 system, will upgrade it to FreeBSD 13.2:

```
# freebsd-update -r 13.2-RELEASE upgrade
```

After the command has been received, `freebsd-update` will evaluate the configuration file and current system in an attempt to gather the information necessary to perform the upgrade. A screen listing will display which components have and have not been detected. For example:

```
Looking up update.FreeBSD.org mirrors... 1 mirrors found.
Fetching metadata signature for 13.1-RELEASE from update1.FreeBSD.org... done.
Fetching metadata index... done.
Inspecting system... done.

The following components of FreeBSD seem to be installed:
kernel/smp src/base src/bin src/contrib src/crypto src/etc src/games
src/gnu src/include src/krb5 src/lib src/libexec src/release src/rescue
src/sbin src/secure src/share src/sys src/tools src/ubin src/usbin
```

```
world/base world/info world/lib32 world/manpages

The following components of FreeBSD do not seem to be installed:
kernel/generic world/catpages world/dict world/doc world/games
world/proflibs

Does this look reasonable (y/n)? y
```

At this point, `freebsd-update` will attempt to download all files required for the upgrade. In some cases, the user may be prompted with questions regarding what to install or how to proceed.

When using a custom kernel, the above step will produce a warning similar to the following:

```
WARNING: This system is running a "MYKERNEL" kernel, which is not a
kernel configuration distributed as part of FreeBSD 13.1-RELEASE.
This kernel will not be updated: you MUST update the kernel manually
before running "/usr/sbin/freebsd-update install"
```

This warning may be safely ignored at this point. The updated GENERIC kernel will be used as an intermediate step in the upgrade process.

Once all the patches have been downloaded to the local system, they will be applied. This process may take a while, depending on the speed and workload of the machine. Configuration files will then be merged. The merging process requires some user intervention as a file may be merged or an editor may appear on screen for a manual merge. The results of every successful merge will be shown to the user as the process continues. A failed or ignored merge will cause the process to abort. Users may wish to make a backup of /etc and manually merge important files, such as master.passwd or group at a later time.

The system is not being altered yet as all patching and merging is happening in another directory. Once all patches have been applied successfully, all configuration files have been merged and it seems the process will go smoothly, the changes can be committed to disk by the user using the following command:

```
# freebsd-update install
```

The kernel and kernel modules will be patched first. If the system is running with a custom kernel, use nextboot(8) to set the kernel for the next boot to the updated /boot/GENERIC:

```
# nextboot -k GENERIC
```

Before rebooting with the GENERIC kernel, make sure it contains all the drivers required for the system to boot properly and connect to the network, if the machine being updated is accessed remotely. In particular, if the running custom kernel contains built-in functionality usually provided by kernel modules, make sure to temporarily load these modules into the GENERIC kernel using the /boot/loader.conf facility. It is recommended to disable non-essential services as well as any disk and network mounts until the upgrade process is complete.

The machine should now be restarted with the updated kernel:

```
# shutdown -r now
```

Once the system has come back online, restart `freebsd-update` using the following command. Since the state of the process has been saved, `freebsd-update` will not start from the beginning, but will instead move on to the next phase and remove all old shared libraries and object files.

```
# freebsd-update install
```

Depending upon whether any library version numbers were bumped, there may only be two install phases instead of three.

The upgrade is now complete. If this was a major version upgrade, reinstall all ports and packages as described in Upgrading Packages After a Major Version Upgrade.

Custom Kernels with FreeBSD 9.X and Later

Before using `freebsd-update`, ensure that a copy of the GENERIC kernel exists in /boot/GENERIC. If a custom kernel has only been built once, the kernel in /boot/kernel.old is the `GENERIC` kernel. Simply rename this directory to /boot/GENERIC.

If a custom kernel has been built more than once or if it is unknown how many times the custom kernel has been built, obtain a copy of the `GENERIC` kernel that matches the current version of the operating system. If physical access to the system is available, a copy of the `GENERIC` kernel can be installed from the installation media:

```
# mount /media
# cd /media/usr/freebsd-dist
# tar -C/ -xvf kernel.txz boot/kernel/kernel
```

Alternately, the `GENERIC` kernel may be rebuilt and installed from source:

```
# cd /usr/src
# make kernel __MAKE_CONF=/dev/null SRCCONF=/dev/null
```

For this kernel to be identified as the `GENERIC` kernel by `freebsd-update`, the GENERIC configuration file must not have been modified in any way. It is also suggested that the kernel is built without any other special options.

Rebooting into the GENERIC kernel is not required as `freebsd-update` only needs /boot/GENERIC to exist.

Upgrading Packages After a Major Version Upgrade

Generally, installed applications will continue to work without problems after minor version upgrades. Major versions use different Application Binary Interfaces (ABIs), which will break most third-party applications. After a major version upgrade, all installed packages and ports need to be upgraded. Packages can be upgraded using `pkg upgrade`. To upgrade installed ports, use a utility such as ports-mgmt/portmaster.

A forced upgrade of all installed packages will replace the packages with fresh versions from the repository even if the version number has not increased. This is required because of the ABI version change when upgrading between major versions of FreeBSD. The forced upgrade can be accomplished by performing:

```
# pkg-static upgrade -f
```

A rebuild of all applications installed from the ports collection can be accomplished with this command:

```
# portmaster -af
```

This command will display the configuration screens for each application that has configurable options and wait for the user to interact with those screens. To prevent this behavior, and use only the default options, include `-G` in the above command.

Once the software upgrades are complete, finish the upgrade process with a final call to `freebsd-update` in order to tie up all the loose ends in the upgrade process:

```
# freebsd-update install
```

If the GENERIC kernel was temporarily used, this is the time to build and install a new custom kernel using the instructions in Configuring the FreeBSD Kernel.

Reboot the machine into the new FreeBSD version. The upgrade process is now complete.

27.2.4. System State Comparison

The state of the installed FreeBSD version against a known good copy can be tested using `freebsd-update IDS`. This command evaluates the current version of system utilities, libraries, and configuration files and can be used as a built-in Intrusion Detection System (IDS).

This command is not a replacement for a real IDS such as security/snort. As `freebsd-update` stores data on disk, the possibility of tampering is evident. While this possibility may be reduced using `kern.securelevel` and by storing the `freebsd-update` data on a read-only file system when not in use, a better solution would be

> to compare the system against a secure disk, such as a DVD or securely stored external USB disk device. An alternative method for providing IDS functionality using a built-in utility is described in Binary Verification

To begin the comparison, specify the output file to save the results to:

```
# freebsd-update IDS >> outfile.ids
```

The system will now be inspected and a lengthy listing of files, along with the SHA256 hash values for both the known value in the release and the current installation, will be sent to the specified output file.

The entries in the listing are extremely long, but the output format may be easily parsed. For instance, to obtain a list of all files which differ from those in the release, issue the following command:

```
# cat outfile.ids | awk '{ print $1 }' | more
/etc/master.passwd
/etc/motd
/etc/passwd
/etc/pf.conf
```

This sample output has been truncated as many more files exist. Some files have natural modifications. For example, /etc/passwd will be modified if users have been added to the system. Kernel modules may differ as `freebsd-update` may have updated them. To exclude specific files or directories, add them to the `IDSIgnorePaths` option in /etc/freebsd-update.conf.

27.3. Updating Bootcode

The following manuals describe the upgrade process of bootcode and boot loaders: gpart(8), gptboot(8), gptzfsboot(8), and loader.efi(8).

27.4. Updating the Documentation Set

Documentation is an integral part of the FreeBSD operating system. While an up-to-date version of the FreeBSD documentation is always available on the FreeBSD web site (Documentation Portal), it can be handy to have an up-to-date, local copy of the FreeBSD website, handbooks, FAQ, and articles.

This section describes how to use either source or the FreeBSD Ports Collection to keep a local copy of the FreeBSD documentation up-to-date.

For information on editing and submitting corrections to the documentation, refer to the FreeBSD Documentation Project Primer for New Contributors (FreeBSD Documentation Project Primer for New Contributors).

27.4.1. Updating Documentation from Source

Rebuilding the FreeBSD documentation from source requires a collection of tools which are not part of the FreeBSD base system. The required tools can be installed following these steps from the FreeBSD Documentation Project Primer.

Once installed, use `git` to fetch a clean copy of the documentation source:

```
# git clone https://git.FreeBSD.org/doc.git /usr/doc
```

The initial download of the documentation sources may take a while. Let it run until it completes.

Future updates of the documentation sources may be fetched by running:

```
# git pull
```

Once an up-to-date snapshot of the documentation sources has been fetched to /usr/doc, everything is ready for an update of the installed documentation.

A full update may be performed by typing:

```
# cd /usr/doc
# make
```

27.5. Tracking a Development Branch

FreeBSD has two development branches: FreeBSD-CURRENT and FreeBSD-STABLE.

This section provides an explanation of each branch and its intended audience, as well as how to keep a system up-to-date with each respective branch.

27.5.1. Using FreeBSD-CURRENT

FreeBSD-CURRENT is the "bleeding edge" of FreeBSD development and FreeBSD-CURRENT users are expected to have a high degree of technical skill. Less technical users who wish to track a development branch should track FreeBSD-STABLE instead.

FreeBSD-CURRENT is the very latest source code for FreeBSD and includes works in progress, experimental changes, and transitional mechanisms that might or might not be present in the next official release. While many FreeBSD developers compile the FreeBSD-CURRENT source code daily, there are short periods of time when the source may not be buildable. These problems are resolved as quickly as possible, but whether or not FreeBSD-CURRENT brings disaster or new functionality can be a matter of when the source code was synced.

FreeBSD-CURRENT is made available for three primary interest groups:

1. Members of the FreeBSD community who are actively working on some part of the source tree.

2. Members of the FreeBSD community who are active testers. They are willing to spend time solving problems, making topical suggestions on changes and the general direction of FreeBSD, and submitting patches.
3. Users who wish to keep an eye on things, use the current source for reference purposes, or make the occasional comment or code contribution.

FreeBSD-CURRENT should *not* be considered a fast-track to getting new features before the next release as pre-release features are not yet fully tested and most likely contain bugs. It is not a quick way of getting bug fixes as any given commit is just as likely to introduce new bugs as to fix existing ones. FreeBSD-CURRENT is not in any way "officially supported".

To track FreeBSD-CURRENT:

1. Join the FreeBSD-CURRENT mailing list and the Commit messages for the main branch of the src repository lists. This is *essential* in order to see the comments that people are making about the current state of the system and to receive important bulletins about the current state of FreeBSD-CURRENT.

   The Commit messages for the main branch of the src repository list records the commit log entry for each change as it is made, along with any pertinent information on possible side effects.

   To join these lists, go to FreeBSD list server, click on the list to subscribe to, and follow the instructions. In order to track changes to the whole source tree, not just the changes to FreeBSD-CURRENT, subscribe to the Commit messages for all branches of the src repository.
2. Synchronize with the FreeBSD-CURRENT sources. Typically, `git` is used to check out the -CURRENT code from the `main` branch of the FreeBSD Git repository (see “Using Git” for details).
3. Due to the size of the repository, some users choose to only synchronize the sections of source that interest them or which they are contributing patches to. However, users that plan to compile the operating system from source must download *all* of FreeBSD-CURRENT, not just selected portions.

   Before compiling FreeBSD-CURRENT, read /usr/src/Makefile very carefully and follow the instructions in Updating FreeBSD from Source. Read the FreeBSD-CURRENT mailing list and /usr/src/UPDATING to stay up-to-date on other bootstrapping procedures that sometimes become necessary on the road to the next release.
4. Be active! FreeBSD-CURRENT users are encouraged to submit their suggestions for enhancements or bug fixes. Suggestions with accompanying code are always welcome.

27.5.2. Using FreeBSD-STABLE

FreeBSD-STABLE is the development branch from which major releases are made. Changes go into this branch at a slower pace and with the general assumption that they have first been tested in FreeBSD-CURRENT. This is *still* a development branch and, at any given time, the sources for FreeBSD-STABLE may or may not be suitable for general use. It is simply another engineering development track, not a resource for end-users. Users who do not have the resources to perform testing should instead run the most recent release of FreeBSD.

Those interested in tracking or contributing to the FreeBSD development process, especially as it relates to the next release of FreeBSD, should consider following FreeBSD-STABLE.

While the FreeBSD-STABLE branch should compile and run at all times, this cannot be guaranteed. Since more people run FreeBSD-STABLE than FreeBSD-CURRENT, it is inevitable that bugs and corner cases will sometimes be found in FreeBSD-STABLE that were not apparent in FreeBSD-CURRENT. For this reason, one should not blindly track FreeBSD-STABLE. It is particularly important *not* to update any production servers to FreeBSD-STABLE without thoroughly testing the code in a development or testing environment.

To track FreeBSD-STABLE:

1. Join the FreeBSD-STABLE mailing list in order to stay informed of build dependencies that may appear in FreeBSD-STABLE or any other issues requiring special attention. Developers will also make announcements in this mailing list when they are contemplating some controversial fix or update, giving the users a chance to respond if they have any issues to raise concerning the proposed change.

   Join the relevant git list for the branch being tracked. For example, users tracking the 14-STABLE branch should join the Commit messages for the stable branches of the src repository. This list records the commit log entry for each change as it is made, along with any pertinent information on possible side effects.

   To join these lists, go to FreeBSD list server, click on the list to subscribe to, and follow the instructions. In order to track changes for the whole source tree, subscribe to Commit messages for all branches of the src repository.

2. To install a new FreeBSD-STABLE system, install the most recent FreeBSD-STABLE release from the FreeBSD mirror sites or use a monthly snapshot built from FreeBSD-STABLE. Refer to www.freebsd.org/snapshots for more information about snapshots.

   To compile or upgrade an existing FreeBSD system to FreeBSD-STABLE, use `git` to check out the source for the desired branch. Branch names, such as `stable/13`, are listed at www.freebsd.org/releng.

3. Before compiling or upgrading to FreeBSD-STABLE , read /usr/src/Makefile carefully and follow the instructions in Updating FreeBSD from Source. Read the FreeBSD-STABLE mailing list and /usr/src/UPDATING to keep up-to-date on other bootstrapping procedures that sometimes become necessary on the road to the next release.

27.5.3. The N-number

When tracking down bugs it is important to know which versions of the source code have been used to create the system exhibiting an issue. FreeBSD provides version information compiled into the kernel. uname(1) retrieves this information, for example:

```
% uname -v
FreeBSD 14.0-CURRENT #112 main-n247514-031260d64c18: Tue Jun 22 20:43:19 MDT 2021
fred@machine:/usr/home/fred/obj/usr/home/fred/git/head/amd64.amd64/sys/FRED
```

The final field gives information regarding the kernel name, the person that built it, and the location that it was compiled in. Looking at the 4th field, it is made up of several parts:

```
main-n247514-031260d64c18

main        ①
n247514     ②
031260d64c18     ③
④
```

① Git branch name. Note: comparisons of n-numbers are only valid on branches published by the project (`main`, `stable/XX` and `releng/XX`). Local branches will have n-numbers that will overlap commits of their parent branch.

② The n-number is a linear count of commits back to the start of the Git repository starting with the Git hash included in the line.

③ Git hash of the checked out tree

④ Sometimes a suffix of `-dirty` is present when the kernel was built in a tree with uncommitted changes. It is absent in this example because the FRED kernel was built from a pristine checkout.

The `git rev-list` command is used to find the n-number corresponding to a Git hash. For example:

```
% git rev-list --first-parent --count 031260d64c18 ①
247514 ②
```

① git hash to translate (the hash from the above example is reused)

② The n-number.

Usually this number is not all that important. However, when bug fixes are committed, this number makes it easy to quickly determine whether the fix is present in the currently running system. Developers will often refer to the hash of the commit (or provide a URL which has that hash), but not the n-number since the hash is the easily visible identifier for a change while the n-number is not. Security advisories and errata notices will also note an n-number, which can be directly compared against the system. When shallow Git clones are used, n-numbers cannot be compared reliably, as the git rev-list command counts all revisions in the repository, which a shallow clone omits.

27.6. Updating FreeBSD from Source

Updating FreeBSD by compiling from source offers several advantages over binary updates. Code can be built with options to take advantage of specific hardware. Parts of the base system can be built with non-default settings, or left out entirely where they are not needed or desired. The build process takes longer to update a system than just installing binary updates, but allows complete customization to produce a tailored version of FreeBSD.

27.6.1. Quick Start

This is a quick reference for the typical steps used to update FreeBSD by building from source. Later sections describe the process in more detail.

When switching from mergemaster(8) to etcupdate(8), the first run might merge changes incorrectly generating spurious conflicts. To prevent this, perform the following steps **before** updating sources and building the new world:

```
# etcupdate extract ①
# etcupdate diff ②
```

① Bootstrap the database of stock /etc files; for more information see etcupdate(8).

② Check the diff after bootstrapping. Trim any local changes that are no longer needed to reduce the chance of conflicts in future updates.

- Update and Build

```
# git -C /usr/src pull ①
check /usr/src/UPDATING  ②
# cd /usr/src           ③
# make -j4 buildworld   ④
# make -j4 kernel       ⑤
# shutdown -r now       ⑥
# etcupdate -p          ⑦
# cd /usr/src           ⑧
# make installworld     ⑨
# etcupdate -B          ⑩
# shutdown -r now       ⑪
```

① Get the latest version of the source. See Updating the Source for more information on obtaining and updating source.

② Check /usr/src/UPDATING for any manual steps required before or after building from source.

③ Go to the source directory.

④ Compile the world, everything except the kernel.

⑤ Compile and install the kernel. This is equivalent to `make buildkernel installkernel`.

⑥ Reboot the system to the new kernel.

⑦ Update and merge configuration files in /etc/ required before installworld.

⑧ Go to the source directory.

⑨ Install the world.

⑩ Update and merge configuration files in /etc/.

⑪ Restart the system to use the newly-built world and kernel.

27.6.2. Preparing for a Source Update

Read /usr/src/UPDATING. Any manual steps that must be performed before or after an update are described in this file.

27.6.3. Updating the Source

FreeBSD source code is located in /usr/src/. The preferred method of updating this source is through the Git version control system. Verify that the source code is under version control:

```
# cd /usr/src
# git remote --v
origin  https://git.freebsd.org/src.git (fetch)
origin  https://git.freebsd.org/src.git (push)
```

This indicates that /usr/src/ is under version control and can be updated with git(1):

```
# git -C /usr/src pull
```

The update process can take some time if the directory has not been updated recently. After it finishes, the source code is up to date and the build process described in the next section can begin.

Obtaining the source:

If the output says `fatal: not a git repository`, the files there are missing or were installed with a different method. A new checkout of the source is required.

Table 39. FreeBSD Versions and Repository Branches

uname -r Output	Repository Path	Description
`X.Y-RELEASE`	`releng/X.Y`	The Release version plus only critical security and bug fix patches. This branch is recommended for most users.
`X.Y-STABLE`	`stable/X`	The Release version plus all additional development on that branch. *STABLE* refers to the Applications Binary Interface (ABI) not changing, so software compiled for earlier versions still runs. For example, software compiled to run on FreeBSD 14.4 will still run on FreeBSD 14-STABLE compiled later. STABLE branches occasionally have bugs or incompatibilities which might affect users, although these are typically fixed quickly.

uname -r Output	Repository Path	Description
`X-CURRENT`	`main`	The latest unreleased development version of FreeBSD. The CURRENT branch can have major bugs or incompatibilities and is recommended only for advanced users.

Determine which version of FreeBSD is being used with uname(1):

```
# uname -r
13.2-RELEASE
```

Based on FreeBSD Versions and Repository Branches, the source used to update `13.2-RELEASE` has a repository path of `releng/13.2`. That path is used when checking out the source:

```
# mv /usr/src /usr/src.bak ①
# git clone --branch releng/13.2 https://git.FreeBSD.org/src.git /usr/src ②
```

① Move the old directory out of the way. If there are no local modifications in this directory, it can be deleted.

② The path from FreeBSD Versions and Repository Branches is added to the repository URL. The third parameter is the destination directory for the source code on the local system.

27.6.4. Building from Source

The *world*, or all of the operating system except the kernel, is compiled. This is done first to provide up-to-date tools to build the kernel. Then the kernel itself is built:

```
# cd /usr/src
# make buildworld
# make buildkernel
```

The compiled code is written to /usr/obj.

These are the basic steps. Additional options to control the build are described below.

Performing a Clean Build

Some versions of the FreeBSD build system leave previously-compiled code in the temporary object directory, /usr/obj. This can speed up later builds by avoiding recompiling code that has not changed. To force a clean rebuild of everything, use `cleanworld` before starting a build:

```
# make cleanworld
```

Setting the Number of Jobs

Increasing the number of build jobs on multi-core processors can improve build speed. Determine the number of cores with `sysctl hw.ncpu`. Processors vary, as do the build systems used with different versions of FreeBSD, so testing is the only sure method to tell how a different number of jobs affects the build speed. For a starting point, consider values between half and double the number of cores. The number of jobs is specified with `-j`.

Example 39. Increasing the Number of Build Jobs

Building the world and kernel with four jobs:

```
# make -j4 buildworld buildkernel
```

Building Only the Kernel

A `buildworld` must be completed if the source code has changed. After that, a `buildkernel` to build a kernel can be run at any time. To build just the kernel:

```
# cd /usr/src
# make buildkernel
```

Building a Custom Kernel

The standard FreeBSD kernel is based on a *kernel config file* called GENERIC. The GENERIC kernel includes the most commonly-needed device drivers and options. Sometimes it is useful or necessary to build a custom kernel, adding or removing device drivers or options to fit a specific need.

For example, someone developing a small embedded computer with severely limited RAM could remove unneeded device drivers or options to make the kernel slightly smaller.

Kernel config files are located in /usr/src/sys/arch/conf/, where *arch* is the output from `uname -m`. On most computers, that is `amd64`, giving a config file directory of /usr/src/sys/amd64/conf/.

/usr/src can be deleted or recreated, so it is preferable to keep custom kernel config files in a separate directory, like /root. Link the kernel config file into the conf directory. If that directory is deleted or overwritten, the kernel config can be re-linked into the new one.

A custom config file can be created by copying the GENERIC config file. In this example, the new custom kernel is for a storage server, so is named STORAGESERVER:

```
# cp /usr/src/sys/amd64/conf/GENERIC /root/STORAGESERVER
# cd /usr/src/sys/amd64/conf
# ln -s /root/STORAGESERVER .
```

/root/STORAGESERVER is then edited, adding or removing devices or options as shown in config(5).

The custom kernel is built by setting `KERNCONF` to the kernel config file on the command line:

```
# make buildkernel KERNCONF=STORAGESERVER
```

27.6.5. Installing the Compiled Code

After the `buildworld` and `buildkernel` steps have been completed, the new kernel and world are installed:

```
# cd /usr/src
# make installkernel
# shutdown -r now
# cd /usr/src
# make installworld
# shutdown -r now
```

If a custom kernel was built, `KERNCONF` must also be set to use the new custom kernel:

```
# cd /usr/src
# make installkernel KERNCONF=STORAGESERVER
# shutdown -r now
# cd /usr/src
# make installworld
# shutdown -r now
```

27.6.6. Completing the Update

A few final tasks complete the update. Any modified configuration files are merged with the new versions, outdated libraries are located and removed, then the system is restarted.

Merging Configuration Files with etcupdate(8)

etcupdate(8) is a tool for managing updates to files that are not updated as part of an installworld such as files located in /etc/. It manages updates by doing a three-way merge of changes made to these files against the local versions. etcupdate(8) is designed to minimize the amount of user intervention.

> In general, etcupdate(8) does not need any specific arguments for its job. There is however a handy in between command for sanity checking what will be done the first time etcupdate(8) is used:
>
> ```
> # etcupdate diff
> ```

This command allows the user to audit configuration changes.

If etcupdate(8) is not able to merge a file automatically, the merge conflicts can be resolved with manual interaction by issuing:

```
# etcupdate resolve
```

When switching from mergemaster(8) to etcupdate(8), the first run might merge changes incorrectly generating spurious conflicts. To prevent this, perform the following steps **before** updating sources and building the new world:

```
# etcupdate extract ①
# etcupdate diff ②
```

① Bootstrap the database of stock /etc files; for more information see etcupdate(8).

② Check the diff after bootstrapping. Trim any local changes that are no longer needed to reduce the chance of conflicts in future updates.

Checking for Outdated Files and Libraries

Some obsolete files or directories can remain after an update. These files can be located:

```
# make check-old
```

and deleted:

```
# make delete-old
```

Some obsolete libraries can also remain. These can be detected with:

```
# make check-old-libs
```

and deleted with

```
# make delete-old-libs
```

Programs which were still using those old libraries will stop working when the library has been deleted. These programs must be rebuilt or replaced after deleting the old libraries.

When all the old files or directories are known to be safe to delete, pressing `y` and `Enter` to delete each file can be avoided by setting `BATCH_DELETE_OLD_FILES` in the command. For example:

```
# make BATCH_DELETE_OLD_FILES=yes delete-old-libs
```

Restarting After the Update

The last step after updating is to restart the computer so all the changes take effect:

```
# shutdown -r now
```

27.7. Updating FreeBSD with packages

Starting from 14.0-RELEASE, the FreeBSD project has published a set of packages of the base system, using pkg(8). The convenience of pkg is no longer limited to the FreeBSD ports collection. pkg(8) can be used for FreeBSD itself. Base packages, and usage thereof, were traditionally referred to as *pkgbase*.

Publication of official packages began in October 2023. Their use with FreeBSD 14 is currently experimental.

Starting from 15.0-RELEASE, freebsd-base(7) will be supported as a tech preview for:

- installation
- minor upgrades
- major upgrades

freebsd-base will eventually replace:

- tarball distribution sets, such as `base.txz` or `kernel.txz`, which are historically used for installation of the OS with bsdinstall(8)
- freebsd-update(8) for updates to the OS.

The current plan is for freebsd-update to work with pkg.

> ⚠ The replacement of distribution sets will take some time, until freebsd-base is fully stable!

Base System packages complement building and installing from source, which is still available for those who wish to build their own custom kernels or userland. It is also possible to build custom base system packages from local sources, as well as just relying on officially provided packages.

27.7.1. Converting a Host to use freebsd-base

FreeBSD 14.0-RELEASE or later can be converted to use base packages. Earlier versions must be upgraded with freebsd-update before conversion.

For conversion, the FreeBSD Foundation-sponsored pkgbasify tool is recommended. Follow the instructions given there.

freebsd-update(8) will be enhanced to use pkg and pkgbasify.

⚠ Note that this migration requires up to 5GiB additional free space, to download, unpack, and relocate any conflicting files. The pkgbasify tool does not check for this and it is the responsibility of the user to ensure that enough space is available before running the migration.

pkgbasify(8) (or whatever outcome D51594 or port request will have) performs 6 main tasks:

- Creates a backup boot environment (ZFS only) with bectl(8)
- Creates the new package repository config files
- Upgrades existing system components such as base, kernel, lib32, debug
- Merges existing and new config files
- Updates passwd and capabilities databases
- Restarts sshd immediately

27.7.2. Upgrading a Host using freebsd-base

⚠ This is still in development, so please be careful especially when converting an existing system to use freebsd-base.

Create a folder for custom pkg repository config files, if there is none present already.

```
# mkdir -p /usr/local/etc/pkg/repos/
```

For using the freebsd-base repository, create a pkg repository configuration file called `FreeBSD-base.conf`:

```
FreeBSD-base {
    url = "pkg+https://pkg.freebsd.org/${ABI}/base_release_${VERSION_MINOR}";
    mirror_type = "srv";
    signature_type = "fingerprints";
    fingerprints = "/usr/share/keys/pkg";
    enabled = yes;
}
```

for more information on specific configuration options see pkg.conf(5).

There are different branches to choose from (by changing the url accordingly):

Table 40. Base system package Branches

Branch	Frequency	URL
main	twice daily - 12:00 and 00:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_latest`

main	weekly – Sunday at 12:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_weekly`
stable/14	twice daily – 12:00 and 00:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_latest`
stable/14	weekly – Sunday at 12:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_weekly`
releng/14.3	twice daily – 12:00 and 00:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_release_3`
releng/15.0	twice daily – 12:00 and 00:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_release_0`
stable/15	twice daily – 12:00 and 00:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_latest`
stable/15	weekly – Sunday at 12:00 UTC	`pkg+https://pkg.freebsd.org/${ABI}/base_weekly`

To upgrade the system, change the configuration file according to the desired release, and run:

```
# pkg update -r FreeBSD-base
# pkg upgrade -r FreeBSD-base
```

check, if these packages are correct and accept the changes.

Reboot the OS executing the following command:

```
# shutdown -r now
```

Major Upgrades

With ZFS, boot environments allow an upgrade to be performed without interrupting running software. If the system does not use root-on-ZFS: backup before upgrading.

Preparations

Change `/usr/local/etc/pkg/repos/FreeBSD-base.conf` to target the correct major release like `base_release_0` for 15.0-RELEASE, to look like this:

This file might change to be included in RELEASE with a different path.

```
FreeBSD-base {
    url = "pkg+https://pkg.freebsd.org/${ABI}/base_release_0";
    mirror_type = "srv";
    signature_type = "fingerprints";
    fingerprints = "/usr/share/keys/pkg";
    enabled = yes;
}
```

As pkgbase with FreeBSD 14 is experimental, so is a pkgbase major upgrade to 15. At the time of this writing in some edge-cases the major upgrade removes pkg and therefore segfaults. This is considered a (known issue) for 15.0-RELEASE. To work around this issue, lock pkg before upgrading.

```
# pkg -c /mnt/upgrade lock pkg
```

An experimental major upgrade from FreeBSD 14, to 15 (or 16.0-CURRENT), should be followed by installation of meta package `FreeBSD-set-minimal`.

Save a list of the non-base packages in case they are needed later:

```
pkg prime-origins | sort -u > /var/tmp/pkg-prime-origins.txt
```

Major Upgrade With ZFS

Create a boot environment using bectl(8) and name it according to the version upgrading to

```
# bectl create 15.0-RELEASE
```

Create a directory then mount the new boot environment there to enable working with pkg chroot(8) and rootdir.

```
# mkdir /mnt/upgrade
# bectl mount 15.0-RELEASE /mnt/upgrade
```

The next step will upgrade the boot environment to the specified version.

This step might remove non-base packages, which could include the running desktop environment.

Set the environment variable ABI to upgrade the major version (replace amd64 with the architecture and 15 with the targeted version). Set the pkg-static chroot directory to the boot environment mountpoint.

```
# env ABI=FreeBSD:15:amd64 pkg-static -c /mnt/upgrade upgrade -r FreeBSD-base
```

There will be a prompt asking about ignoring the version mismatch looking like this:

```
Newer FreeBSD version for package FreeBSD-zoneinfo:
To ignore this error set IGNORE_OSVERSION=yes
```

```
- package: 1500058
- running userland: 1500000
Ignore the mismatch and continue? [y/N]:
```

Check and confirm that.

To check if that was successful, chroot into the mountpoint of the boot environment and run `freebsd-version -ku`.

```
# chroot /mnt/upgrade
# freebsd-version -ku
# exit
```

Activate the boot environment temporarily and reboot.

```
# bectl activate -t 15.0-RELEASE
# shutdown -r now
```

If the system does not work as expected: restart the OS to cease using the temporarily active boot environment. The next boot will reuse the pre-upgrade environment.

If the system runs stable after reboot, don’t forget to permanently activate this boot environment to prevent accidentally booting the old system.

```
bectl activate 15.0-RELEASE
```

Major Upgrade Without ZFS

Set the environment variable ABI to upgrade the major version (replace amd64 with the architecture and 15 with the targeted version).

```
# env ABI=FreeBSD:15:amd64 pkg-static upgrade -r FreeBSD-base
```

There will be a prompt to ask about ignoring the version mismatch looking like this:

```
Newer FreeBSD version for package FreeBSD-zoneinfo:
To ignore this error set IGNORE_OSVERSION=yes
- package: 1500058
- running userland: 1500000
Ignore the mismatch and continue? [y/N]:
```

Check and confirm that.

To check if the upgrade was successful, run `freebsd-version -ku`.

Then reboot.

Post-upgrade tasks

If pkg was locked during the update as a workaround remove the lock like this:

```
# pkg unlock pkg
```

After upgrading to a new major version, updates and upgrades of installed packages to match the ABI version may be necessary.

```
# pkg upgrade
```

Consider getting help from FreeBSD Support when experiencing issues.

27.7.3. Manually building pkgbase and publishing it to the local network

When building custom pkgbase packages clone the FreeBSD source tree:

```
# cd /usr/src
# git clone https://github.com/freebsd/freebsd-src.git /usr/src
```

Check out the branch for the release to build packages for:

```
# git checkout releng/14.3
```

Start the building process, depending on the resources available this could take some while. Set the parallel processes according to the CPU core count.

This example is written for an 8 core CPU:

```
# make -j8 buildworld && make -j8 buildkernel && make -j8 packages
```

When building frequently, consider using devel/ccache to speed up subsequent builds from the cache.

After building, the packages will get saved into `/usr/obj/usr/src/repo/FreeBSD:14:amd64/14.3p2` or something like that, depending on the version build.

To publish these packages to the network set up a nginx service and use this location in the http server configuration:

```
  location /FreeBSD:14:amd64 {
    alias /usr/obj/usr/src/repo/FreeBSD:14:amd64/;
    autoindex on;
  }
```

And reload the nginx service.

When not using https, use a small configuration file on the clients to target the pkgbase version just built by editing `/usr/local/etc/pkg/repos/upgrade.conf`:

```
upgrade: {
    url         = http://ip.of.the.server/FreeBSD:14:amd64/14.3p2
    enabled     = yes
}
```

and use it as written above (but use -r upgrade instead of FreeBSD-base).

27.8. Tracking for Multiple Machines

When multiple machines need to track the same source tree, it is a waste of disk space, network bandwidth, and CPU cycles to have each system download the sources and rebuild everything. The solution is to have one machine do most of the work, while the rest of the machines mount that work via NFS. This section outlines a method of doing so. For more information about using NFS, refer to Network File System (NFS).

First, identify a set of machines which will run the same set of binaries, known as a *build set*. Each machine can have a custom kernel, but will run the same userland binaries. From that set, choose a machine to be the *build machine* that the world and kernel are built on. Ideally, this is a fast machine that has sufficient spare CPU to run `make buildworld` and `make buildkernel`.

Select a machine to be the *test machine*, which will test software updates before they are put into production. This *must* be a machine that can afford to be down for an extended period of time. It can be the build machine, but need not be.

All the machines in this build set need to mount /usr/obj and /usr/src from the build machine via NFS. For multiple build sets, /usr/src should be on one build machine, and NFS mounted on the rest.

Ensure that /etc/make.conf and /etc/src.conf on all the machines in the build set agree with the build machine. That means that the build machine must build all the parts of the base system that any machine in the build set is going to install. Also, each build machine should have its kernel name set with `KERNCONF` in /etc/make.conf, and the build machine should list them all in its `KERNCONF`, listing its own kernel first. The build machine must have the kernel configuration files for each machine in its /usr/src/sys/arch/conf.

On the build machine, build the kernel and world as described in Updating FreeBSD from Source, but do not install anything on the build machine. Instead, install the built kernel on the test machine. On the test machine, mount /usr/src and /usr/obj via NFS. Then, run `shutdown now` to go to single-user mode in order to install the new kernel and world and run `etcupdate` as usual. When done, reboot to return to normal multi-user operations.

After verifying that everything on the test machine is working properly, use the same procedure to install the new software on each of the other machines in the build set.

The same methodology can be used for the ports tree. The first step is to share /usr/ports via NFS to all the machines in the build set. To configure /etc/make.conf to share distfiles, set `DISTDIR` to a common shared directory that is writable by whichever user `root` is mapped to by the NFS mount. Each machine should set `WRKDIRPREFIX` to a local build directory, if ports are to be built locally. Alternately, if the build system is to build and distribute packages to the machines in the build set, set `PACKAGES` on the build system to a directory similar to `DISTDIR`.

27.9. Building on non-FreeBSD Hosts

Historically, building required a FreeBSD host. Nowadays, FreeBSD can be built on Linux and macOS.

To build on a non-FreeBSD host, the `tools/build/make.py` script is recommended. This script acts as a wrapper around `bmake`, which is the make implementation used by FreeBSD. It ensures that the necessary tooling, including the actual FreeBSD's make(1), is bootstrapped and that the build environment is properly configured. In particular, it sets the external toolchain variables, such as `XCC`, `XLD`, and others. Additionally, the script can pass any additional command arguments, such as `-j 4` for parallel builds or specific make targets, to `bmake`.

> A recent version of `bmake` can be used instead of the `tools/build/make.py` script as well. In that case, however, required environment variables need to be set manually (the easiest way to obtain a list of them is by running `tools/build/make.py --debug`).

Otherwise, the list of prerequisites for building FreeBSD is rather short. In fact, it boils down to installing a couple of dependencies.

On macOS, the only dependency is LLVM. The necessary dependencies can be installed with package manager (e.g., Homebrew):

```
brew install llvm
```

On a Linux distributions, install Clang version 10.0 or newer and the headers for libarchive and libbz2 (often packaged as libarchive-dev and libbz2-dev).

Once the dependencies are installed, the host should be able to build FreeBSD.

For example, the following `tools/build/make.py` invocation builds the world:

```
MAKEOBJDIRPREFIX=/tmp/obj tools/build/make.py -j 8 TARGET=arm64 TARGET_ARCH=aarch64
buildworld
```

It builds the world for target `aarch64:arm64` on 8 CPUs and uses /tmp/obj for object files. Note that the variables `MAKEOBJDIRPREFIX`, `TARGET`, and `TARGET_ARCH` are mandatory when building on non-

FreeBSD hosts. Also, make sure to create the object directory pointed to by the `MAKEOBJDIRPREFIX` environment variable.

Refer to arch(7) and build(7) for more details.

Chapter 28. DTrace

28.1. Synopsis

DTrace, also known as Dynamic Tracing, was developed by Sun™ as a tool for locating performance bottlenecks in production and pre-production systems. In addition to diagnosing performance problems, DTrace can be used to help investigate and debug unexpected behavior in both the FreeBSD kernel and in userland programs.

DTrace is a remarkable profiling tool, with an impressive array of features for diagnosing system issues. It may also be used to run pre-written scripts to take advantage of its capabilities. Users can author their own utilities using the DTrace D Language, allowing them to customize their profiling based on specific needs.

The FreeBSD implementation provides full support for kernel DTrace and experimental support for userland DTrace. Userland DTrace allows users to perform function boundary tracing for userland programs using the `pid` provider, and to insert static probes into userland programs for later tracing. Some ports, such as databases/postgresql12-server and lang/php74 have a DTrace option to enable static probes.

The official guide to DTrace is maintained by the illumos project at illumos Dynamic Tracing Guide.

Read this chapter to learn:

- What DTrace is and what features it provides.
- Differences between the Solaris™ DTrace implementation and the one provided by FreeBSD.
- How to enable and use DTrace on FreeBSD.

Before reading this chapter:

- Understand UNIX® and FreeBSD basics (FreeBSD Basics).
- Have some familiarity with security and how it pertains to FreeBSD (Security).

28.2. Implementation Differences

While the DTrace in FreeBSD is similar to that found in Solaris™, differences do exist. The primary difference is that in FreeBSD, DTrace is implemented as a set of kernel modules and DTrace can not be used until the modules are loaded. To load all of the necessary modules:

```
# kldload dtraceall
```

Beginning with FreeBSD 10.0-RELEASE, the modules are automatically loaded when dtrace(1) is run.

FreeBSD uses the `DDB_CTF` kernel option to enable support for loading ctf(5) data from kernel modules and the kernel itself. `CTF` is the Solaris™ Compact C Type Format which encapsulates a reduced form of debugging information similar to `DWARF` and the venerable stabs. `CTF` data is added to binaries by the ctfconvert(1) and ctfmerge(1) build tools. The `ctfconvert` utility parses `DWARF` `ELF` debug sections created by the compiler and `ctfmerge` merges `CTF` `ELF` sections from objects into either executables or shared libraries.

Some different providers exist for FreeBSD than for Solaris™. Most notable is the `dtmalloc` provider, which allows tracing malloc(9) by type in the FreeBSD kernel. Some of the providers found in Solaris™, such as `cpc`, are not present in FreeBSD. These may appear in future versions of FreeBSD. Moreover, some of the providers available in both operating systems are not compatible, in the sense that their probes have different argument types. Thus, `D` scripts written on Solaris™ may or may not work unmodified on FreeBSD, and vice versa.

Due to security differences, only `root` may use DTrace on FreeBSD. Solaris™ has a few low level security checks which do not yet exist in FreeBSD. As such, the /dev/dtrace/dtrace is strictly limited to `root`.

DTrace falls under the Common Development and Distribution License (`CDDL`) license. To view this license on FreeBSD, see /usr/src/cddl/contrib/opensolaris/OPENSOLARIS.LICENSE or view it online at http://opensource.org/licenses/CDDL-1.0. While a FreeBSD kernel with DTrace support is `BSD` licensed, the `CDDL` is used when the modules are distributed in binary form or the binaries are loaded.

28.3. Enabling DTrace Support

In FreeBSD 9.2 and 10.0, DTrace support is built into the GENERIC kernel. Users of earlier versions of FreeBSD or who prefer to statically compile in DTrace support should add the following lines to a custom kernel configuration file and recompile the kernel using the instructions in Configuring the FreeBSD Kernel:

```
options         KDTRACE_HOOKS
options         DDB_CTF
makeoptions DEBUG=-g
makeoptions WITH_CTF=1
```

Users of the AMD64 architecture should also add this line:

```
options         KDTRACE_FRAME
```

This option provides support for dtrace_fbt(4). While DTrace will work without this option, there will be limited support for function boundary tracing.

Once the FreeBSD system has rebooted into the new kernel, or the DTrace kernel modules have been loaded using `kldload dtraceall`, install the current DTrace Toolkit (sysutils/dtrace-toolkit), a collection of ready-made scripts for collecting system information. There are scripts to check open files, memory, CPU usage, and a lot more. FreeBSD includes some scripts in the base system as well; see /usr/share/dtrace.

The scripts found in /usr/share/dtrace have been specifically ported to FreeBSD. Not all of the scripts found in the DTrace Toolkit will work as-is on FreeBSD and some scripts may require some effort in order for them to work on FreeBSD.

The DTrace Toolkit includes many scripts in the special language of DTrace. This language is called the D language and it is very similar to C++. An in depth discussion of the language is beyond the scope of this document. Refer to the FreeBSD d(7) manual page for an overview of the D language on FreeBSD. The D language is also covered extensively in the illumos Dynamic Tracing Guide.

28.4. Enabling DTrace in Out-of-Kernel Modules

To add DTrace support to an out-of-kernel module, which is useful for development and debugging, include the following line in the module's Makefile:

```
CFLAGS+= -DKDTRACE_HOOKS
```

This flag enables DTrace hooks during compilation, allowing for advanced debugging and monitoring of the module. Ensure to recompile the module after this modification to activate DTrace functionality.

28.5. Using DTrace

DTrace scripts consist of a list of one or more *probes*, or instrumentation points, where each probe is associated with an action. Whenever the condition for a probe is met, the associated action is executed. For example, an action may occur when a file is opened, a process is started, or a line of code is executed. The action might be to log some information or to modify context variables. The reading and writing of context variables allows probes to share information and to cooperatively analyze the correlation of different events.

To view all probes, the administrator can execute the following command:

```
# dtrace -l | more
```

Each probe has an `ID`, a `PROVIDER` (e.g., `dtrace` or `fbt`), a `MODULE`, and a `FUNCTION NAME`. Refer to dtrace(1) for more information about this command.

The examples in this section provide an overview of how to use two of the fully supported scripts from the DTrace Toolkit: the hotkernel and procsystime scripts.

The hotkernel script is designed to identify which function is using the most kernel time. It will produce output similar to the following:

```
# cd /usr/local/share/dtrace-toolkit
# ./hotkernel
Sampling... Hit Ctrl-C to end.
```

As instructed, use the `Ctrl` + `C` key combination to stop the process. Upon termination, the script will display a list of kernel functions and timing information, sorting the output in increasing order of time:

```
kernel`_thread_lock_flags                                   2   0.0%
0xc1097063                                                  2   0.0%
kernel`sched_userret                                        2   0.0%
kernel`kern_select                                          2   0.0%
kernel`generic_copyin                                       3   0.0%
kernel`_mtx_assert                                          3   0.0%
kernel`vm_fault                                             3   0.0%
kernel`sopoll_generic                                       3   0.0%
kernel`fixup_filename                                       4   0.0%
kernel`_isitmyx                                             4   0.0%
kernel`find_instance                                        4   0.0%
kernel`_mtx_unlock_flags                                    5   0.0%
kernel`syscall                                              5   0.0%
kernel`DELAY                                                5   0.0%
0xc108a253                                                  6   0.0%
kernel`witness_lock                                         7   0.0%
kernel`read_aux_data_no_wait                                7   0.0%
kernel`Xint0x80_syscall                                     7   0.0%
kernel`witness_checkorder                                   7   0.0%
kernel`sse2_pagezero                                        8   0.0%
kernel`strncmp                                              9   0.0%
kernel`spinlock_exit                                       10   0.0%
kernel`_mtx_lock_flags                                     11   0.0%
kernel`witness_unlock                                      15   0.0%
kernel`sched_idletd                                       137   0.3%
0xc10981a5                                              42139  99.3%
```

This script will also work with kernel modules. To use this feature, run the script with `-m`:

```
# ./hotkernel -m
Sampling... Hit Ctrl-C to end.
^C
MODULE                                                  COUNT   PCNT
0xc107882e                                                  1   0.0%
0xc10e6aa4                                                  1   0.0%
0xc1076983                                                  1   0.0%
0xc109708a                                                  1   0.0%
0xc1075a5d                                                  1   0.0%
0xc1077325                                                  1   0.0%
0xc108a245                                                  1   0.0%
0xc107730d                                                  1   0.0%
0xc1097063                                                  2   0.0%
0xc108a253                                                 73   0.0%
kernel                                                    874   0.4%
```

```
0xc10981a5                                                  213781  99.6%
```

The procsystime script captures and prints the system call time usage for a given process `ID` (`PID`) or process name. In the following example, a new instance of /bin/csh was spawned. Then, procsystime was executed and remained waiting while a few commands were typed on the other incarnation of `csh`. These are the results of this test:

```
# ./procsystime -n csh
Tracing... Hit Ctrl-C to end...
^C

Elapsed Times for processes csh,

         SYSCALL          TIME (ns)
          getpid               6131
       sigreturn               8121
           close              19127
           fcntl              19959
             dup              26955
         setpgid              28070
            stat              31899
       setitimer              40938
           wait4              62717
       sigaction              67372
     sigprocmask             119091
     gettimeofday            183710
           write             263242
          execve             492547
           ioctl             770073
           vfork            3258923
      sigsuspend            6985124
            read         3988049784
```

As shown, the read(2) system call used the most time in nanoseconds while the getpid(2) system call used the least amount of time.

Chapter 29. USB Device Mode / USB OTG

29.1. Synopsis

This chapter covers the use of USB Device Mode and USB On The Go (USB OTG) in FreeBSD. This includes virtual serial consoles, virtual network interfaces, and virtual USB drives.

When running on hardware that supports USB device mode or USB OTG, like that built into many embedded boards, the FreeBSD USB stack can run in *device mode*. Device mode makes it possible for the computer to present itself as different kinds of USB device classes, including serial ports, network adapters, and mass storage, or a combination thereof. A USB host like a laptop or desktop computer is able to access them just like physical USB devices. Device mode is sometimes called the "USB gadget mode".

There are two basic ways the hardware can provide the device mode functionality: with a separate "client port", which only supports the device mode, and with a USB OTG port, which can provide both device and host mode. For USB OTG ports, the USB stack switches between host-side and device-side automatically, depending on what is connected to the port. Connecting a USB device like a memory stick to the port causes FreeBSD to switch to host mode. Connecting a USB host like a computer causes FreeBSD to switch to device mode. Single purpose "client ports" always work in device mode.

What FreeBSD presents to the USB host depends on the `hw.usb.template` sysctl. Some templates provide a single device, such as a serial terminal; others provide multiple ones, which can all be used at the same time. An example is the template 10, which provides a mass storage device, a serial console, and a network interface. See usb_template(4) for the list of available values.

Note that in some cases, depending on the hardware and the hosts operating system, for the host to notice the configuration change, it must be either physically disconnected and reconnected, or forced to rescan the USB bus in a system-specific way. When FreeBSD is running on the host, usbconfig(8) `reset` can be used. This also must be done after loading usb_template.ko if the USB host was already connected to the USBOTG socket.

Read this chapter to learn:

- How to set up USB Device Mode functionality on FreeBSD.
- How to configure the virtual serial port on FreeBSD.
- How to connect to the virtual serial port from various operating systems.
- How to configure FreeBSD to provide a virtual USB network interface.
- How to configure FreeBSD to provide a virtual USB storage device.

29.2. USB Virtual Serial Ports

29.2.1. Configuring USB Device Mode Serial Ports

Virtual serial port support is provided by templates number 3, 8, and 10. Note that template 3 works with Microsoft Windows 10 without the need for special drivers and INF files. Other host operating systems work with all three templates. Both usb_template(4) and umodem(4) kernel modules must be loaded.

To enable USB device mode serial ports, add those lines to /etc/ttys:

```
ttyU0   "/usr/libexec/getty 3wire"  vt100   onifconsole secure
ttyU1   "/usr/libexec/getty 3wire"  vt100   onifconsole secure
```

Then add these lines to /etc/devd.conf:

```
notify 100 {
    match "system"      "DEVFS";
    match "subsystem"   "CDEV";
    match "type"        "CREATE";
    match "cdev"        "ttyU[0-9]+";
    action "/sbin/init q";
};
```

Reload the configuration if devd(8) is already running:

```
# service devd restart
```

Make sure the necessary modules are loaded and the correct template is set at boot by adding those lines to /boot/loader.conf, creating it if it does not already exist:

```
umodem_load="YES"
hw.usb.template=3
```

To load the module and set the template without rebooting use:

```
# kldload umodem
# sysctl hw.usb.template=3
```

29.2.2. Connecting to USB Device Mode Serial Ports from FreeBSD

To connect to a board configured to provide USB device mode serial ports, connect the USB host, such as a laptop, to the boards USB OTG or USB client port. Use `pstat -t` on the host to list the terminal lines. Near the end of the list a USB serial port is shown, e.g. "ttyU0". To open the connection, use:

```
# cu -l /dev/ttyU0
```

After pressing the `Enter` key a few times a login prompt appears.

29.2.3. Connecting to USB Device Mode Serial Ports from macOS

To connect to a board configured to provide USB device mode serial ports, connect the USB host, such as a laptop, to the boards USB OTG or USB client port. To open the connection, use:

```
# cu -l /dev/cu.usbmodemFreeBSD1
```

29.2.4. Connecting to USB Device Mode Serial Ports from Linux

To connect to a board configured to provide USB device mode serial ports, connect the USB host, such as a laptop, to the boards USB OTG or USB client port. To open the connection, use:

```
# minicom -D /dev/ttyACM0
```

29.2.5. Connecting to USB Device Mode Serial Ports from Microsoft Windows 10

To connect to a board configured to provide USB device mode serial ports, connect the USB host, such as a laptop, to the boards USB OTG or USB client port. To open a connection a serial terminal program is needed, such as PuTTY. To check the COM port name used by Windows, run Device Manager, expand "Ports (COM & LPT)". A name similar to "USB Serial Device (COM4)" appears. Run the serial terminal program of choice, for example PuTTY. In the PuTTY dialog set "Connection type" to "Serial", type the COMx obtained from Device Manager in the "Serial line" dialog box and click Open.

29.3. USB Device Mode Network Interfaces

Virtual network interfaces support is provided by templates number 1, 8, and 10. Note that none of them works with Microsoft Windows. Other host operating systems work with all three templates. Both usb_template(4) and if_cdce(4) kernel modules must be loaded.

Make sure the necessary modules are loaded and the correct template is set at boot by adding those lines to /boot/loader.conf, creating it if it does not already exist:

```
if_cdce_load="YES"
hw.usb.template=1
```

To load the module and set the template without rebooting use:

```
# kldload if_cdce
# sysctl hw.usb.template=1
```

29.4. USB Virtual Storage Device

The cfumass(4) driver is a USB device mode driver first available in FreeBSD 12.0.

Mass Storage target is provided by templates 0 and 10. Both usb_template(4) and cfumass(4) kernel modules must be loaded. cfumass(4) interfaces to the CTL subsystem, the same one that is used for iSCSI or Fibre Channel targets. On the host side, USB Mass Storage initiators can only access a single LUN, LUN 0.

29.4.1. Configuring USB Mass Storage Target Using the cfumass Startup Script

The simplest way to set up a read-only USB storage target is to use the cfumass rc script. To configure it this way, copy the files to be presented to the USB host machine into the `/var/cfumass` directory, and add this line to /etc/rc.conf:

```
cfumass_enable="YES"
```

To configure the target without restarting, run this command:

```
# service cfumass start
```

Differently from serial and network functionality, the template should not be set to 0 or 10 in /boot/loader.conf. This is because the LUN must be set up before setting the template. The cfumass startup script sets the correct template number automatically when started.

29.4.2. Configuring USB Mass Storage Using Other Means

The rest of this chapter provides detailed description of setting the target without using the cfumass rc file. This is necessary if e.g. one wants to provide a writeable LUN.

USB Mass Storage does not require the ctld(8) daemon to be running, although it can be used if desired. This is different from iSCSI. Thus, there are two ways to configure the target: ctladm(8), or ctld(8). Both require the cfumass.ko kernel module to be loaded. The module can be loaded manually:

```
# kldload cfumass
```

If cfumass.ko has not been built into the kernel, /boot/loader.conf can be set to load the module at boot:

```
cfumass_load="YES"
```

A LUN can be created without the ctld(8) daemon:

```
# ctladm create -b block -o file=/data/target0
```

This presents the contents of the image file /data/target0 as a LUN to the USB host. The file must exist before executing the command. To configure the LUN at system startup, add the command to /etc/rc.local.

ctld(8) can also be used to manage LUNs. Create /etc/ctl.conf, add a line to /etc/rc.conf to make sure ctld(8) is automatically started at boot, and then start the daemon.

This is an example of a simple /etc/ctl.conf configuration file. Refer to ctl.conf(5) for a complete description of the options.

```
target naa.50015178f369f092 {
    lun 0 {
        path /data/target0
        size 4G
    }
}
```

The example creates a single target with a single LUN. The `naa.50015178f369f092` is a device identifier composed of 32 random hexadecimal digits. The `path` line defines the full path to a file or zvol backing the LUN. That file must exist before starting ctld(8). The second line is optional and specifies the size of the LUN.

To make sure the ctld(8) daemon is started at boot, add this line to /etc/rc.conf:

```
ctld_enable="YES"
```

To start ctld(8) now, run this command:

```
# service ctld start
```

As the ctld(8) daemon is started, it reads /etc/ctl.conf. If this file is edited after the daemon starts, reload the changes so they take effect immediately:

```
# service ctld reload
```

Part IV: Network Communication

FreeBSD is one of the most widely deployed operating systems for high performance network servers. The chapters in this part cover:

- Serial communication
- `PPP` and `PPP` over Ethernet
- Electronic Mail
- Running Network Servers
- Firewalls
- Other Advanced Networking Topics

These chapters are designed to be read when the information is needed. They do not need to be read in any particular order, nor is it necessary to read all of them before using FreeBSD in a network environment.

Chapter 30. Serial Communications

30.1. Synopsis

UNIX® has always had support for serial communications as the very first UNIX® machines relied on serial lines for user input and output. Things have changed a lot from the days when the average terminal consisted of a 10-character-per-second serial printer and a keyboard. This chapter covers some of the ways serial communications can be used on FreeBSD.

Read this chapter to learn:

- How to connect terminals to a FreeBSD system.
- How to use a modem to dial out to remote hosts.
- How to allow remote users to login to a FreeBSD system with a modem.
- How to boot a FreeBSD system from a serial console.

Before reading this chapter:

- Know how to configure and install a custom kernel.
- Understand FreeBSD permissions and processes.
- Have access to the technical manual for the serial hardware to be used with FreeBSD.

30.2. Serial Terminology and Hardware

The following terms are often used in serial communications:

bps

Bits per Second (bps) is the rate at which data is transmitted.

DTE

Data Terminal Equipment (DTE) is one of two endpoints in a serial communication. An example would be a computer.

DCE

Data Communications Equipment (DCE) is the other endpoint in a serial communication. Typically, it is a modem or serial terminal.

RS-232

The original standard which defined hardware serial communications. It has since been renamed to TIA-232.

When referring to communication data rates, this section does not use the term *baud*. Baud refers to the number of electrical state transitions made in a period of time, while bps is the correct term to use.

To connect a serial terminal to a FreeBSD system, a serial port on the computer and the proper cable to connect to the serial device are needed. Users who are already familiar with serial hardware and cabling can safely skip this section.

30.2.1. Serial Cables and Ports

There are several different kinds of serial cables. The two most common types are null-modem cables and standard RS-232 cables. The documentation for the hardware should describe the type of cable required.

These two types of cables differ in how the wires are connected to the connector. Each wire represents a signal, with the defined signals summarized in RS-232C Signal Names. A standard serial cable passes all of the RS-232C signals straight through. For example, the "Transmitted Data" pin on one end of the cable goes to the "Transmitted Data" pin on the other end. This is the type of cable used to connect a modem to the FreeBSD system, and is also appropriate for some terminals.

A null-modem cable switches the "Transmitted Data" pin of the connector on one end with the "Received Data" pin on the other end. The connector can be either a DB-25 or a DB-9.

A null-modem cable can be constructed using the pin connections summarized in DB-25 to DB-25 Null-Modem Cable, DB-9 to DB-9 Null-Modem Cable, and DB-9 to DB-25 Null-Modem Cable. While the standard calls for a straight-through pin 1 to pin 1 "Protective Ground" line, it is often omitted. Some terminals work using only pins 2, 3, and 7, while others require different configurations. When in doubt, refer to the documentation for the hardware.

Table 41. RS-232C Signal Names

Acronyms	Names
RD	Received Data
TD	Transmitted Data
DTR	Data Terminal Ready
DSR	Data Set Ready
DCD	Data Carrier Detect
SG	Signal Ground
RTS	Request to Send
CTS	Clear to Send

Table 42. DB-25 to DB-25 Null-Modem Cable

Signal	Pin #		Pin #	Signal
SG	7	connects to	7	SG
TD	2	connects to	3	RD
RD	3	connects to	2	TD
RTS	4	connects to	5	CTS
CTS	5	connects to	4	RTS

Signal	Pin #		Pin #	Signal
DTR	20	connects to	6	DSR
DTR	20	connects to	8	DCD
DSR	6	connects to	20	DTR
DCD	8	connects to	20	DTR

Table 43. DB-9 to DB-9 Null-Modem Cable

Signal	Pin #		Pin #	Signal
RD	2	connects to	3	TD
TD	3	connects to	2	RD
DTR	4	connects to	6	DSR
DTR	4	connects to	1	DCD
SG	5	connects to	5	SG
DSR	6	connects to	4	DTR
DCD	1	connects to	4	DTR
RTS	7	connects to	8	CTS
CTS	8	connects to	7	RTS

Table 44. DB-9 to DB-25 Null-Modem Cable

Signal	Pin #		Pin #	Signal
RD	2	connects to	2	TD
TD	3	connects to	3	RD
DTR	4	connects to	6	DSR
DTR	4	connects to	8	DCD
SG	5	connects to	7	SG
DSR	6	connects to	20	DTR
DCD	1	connects to	20	DTR
RTS	7	connects to	5	CTS
CTS	8	connects to	4	RTS

When one pin at one end connects to a pair of pins at the other end, it is usually implemented with one short wire between the pair of pins in their connector and a long wire to the other single pin.

Serial ports are the devices through which data is transferred between the FreeBSD host computer and the terminal. Several kinds of serial ports exist. Before purchasing or constructing a cable, make sure it will fit the ports on the terminal and on the FreeBSD system.

Most terminals have DB-25 ports. Personal computers may have DB-25 or DB-9 ports. A multiport serial card may have RJ-12 or RJ-45/ ports. See the documentation that accompanied the hardware for specifications on the kind of port or visually verify the type of port.

In FreeBSD, each serial port is accessed through an entry in /dev. There are two different kinds of entries:

- Call-in ports are named /dev/ttyuN where *N* is the port number, starting from zero. If a terminal is connected to the first serial port (COM1), use /dev/ttyu0 to refer to the terminal. If the terminal is on the second serial port (COM2), use /dev/ttyu1, and so forth. Generally, the call-in port is used for terminals. Call-in ports require that the serial line assert the "Data Carrier Detect" signal to work correctly.
- Call-out ports are named /dev/cuauN on FreeBSD versions 8.X and higher and /dev/cuadN on FreeBSD versions 7.X and lower. Call-out ports are usually not used for terminals, but are used for modems. The call-out port can be used if the serial cable or the terminal does not support the "Data Carrier Detect" signal.

FreeBSD also provides initialization devices (/dev/ttyuN.init and /dev/cuauN.init or /dev/cuadN.init) and locking devices (/dev/ttyuN.lock and /dev/cuauN.lock or /dev/cuadN.lock). The initialization devices are used to initialize communications port parameters each time a port is opened, such as `crtscts` for modems which use `RTS/CTS` signaling for flow control. The locking devices are used to lock flags on ports to prevent users or programs changing certain parameters. Refer to termios(4), uart(4), and stty(1) for information on terminal settings, locking and initializing devices, and setting terminal options, respectively.

30.2.2. Serial Port Configuration

By default, FreeBSD supports four serial ports which are commonly known as COM1, COM2, COM3, and COM4. FreeBSD also supports dumb multi-port serial interface cards, such as the BocaBoard 1008 and 2016, as well as more intelligent multi-port cards such as those made by Digiboard. However, the default kernel only looks for the standard COM ports.

To see if the system recognizes the serial ports, look for system boot messages that start with `uart`:

```
# grep uart /var/run/dmesg.boot
```

If the system does not recognize all of the needed serial ports, additional entries can be added to /boot/device.hints. This file already contains `hint.uart.0.*` entries for COM1 and `hint.uart.1.*` entries for COM2. When adding a port entry for COM3 use `0x3E8`, and for COM4 use `0x2E8`. Common IRQ addresses are `5` for COM3 and `9` for COM4.

To determine the default set of terminal I/O settings used by the port, specify its device name. This example determines the settings for the call-in port on COM2:

```
# stty -a -f /dev/ttyu1
```

System-wide initialization of serial devices is controlled by /etc/rc.d/serial. This file affects the default settings of serial devices. To change the settings for a device, use `stty`. By default, the changed settings are in effect until the device is closed and when the device is reopened, it goes back to the default set. To permanently change the default set, open and adjust the settings of the initialization device. For example, to turn on `CLOCAL` mode, 8 bit communication, and `XON/XOFF` flow control for ttyu5, type:

```
# stty -f /dev/ttyu5.init clocal cs8 ixon ixoff
```

To prevent certain settings from being changed by an application, make adjustments to the locking device. For example, to lock the speed of ttyu5 to 57600 bps, type:

```
# stty -f /dev/ttyu5.lock 57600
```

Now, any application that opens ttyu5 and tries to change the speed of the port will be stuck with 57600 bps.

30.3. Terminals

Terminals provide a convenient and low-cost way to access a FreeBSD system when not at the computer's console or on a connected network. This section describes how to use terminals with FreeBSD.

The original UNIX® systems did not have consoles. Instead, users logged in and ran programs through terminals that were connected to the computer's serial ports.

The ability to establish a login session on a serial port still exists in nearly every UNIX®-like operating system today, including FreeBSD. By using a terminal attached to an unused serial port, a user can log in and run any text program that can normally be run on the console or in an `xterm` window.

Many terminals can be attached to a FreeBSD system. An older spare computer can be used as a terminal wired into a more powerful computer running FreeBSD. This can turn what might otherwise be a single-user computer into a powerful multiple-user system.

FreeBSD supports three types of terminals:

Dumb terminals

Dumb terminals are specialized hardware that connect to computers over serial lines. They are called "dumb" because they have only enough computational power to display, send, and receive text. No programs can be run on these devices. Instead, dumb terminals connect to a computer that runs the needed programs.

There are hundreds of kinds of dumb terminals made by many manufacturers, and just about any kind will work with FreeBSD. Some high-end terminals can even display graphics, but only certain software packages can take advantage of these advanced features.

Dumb terminals are popular in work environments where workers do not need access to graphical applications.

Computers Acting as Terminals

Since a dumb terminal has just enough ability to display, send, and receive text, any spare computer can be a dumb terminal. All that is needed is the proper cable and some *terminal emulation* software to run on the computer.

This configuration can be useful. For example, if one user is busy working at the FreeBSD system's console, another user can do some text-only work at the same time from a less powerful personal computer hooked up as a terminal to the FreeBSD system.

There are at least two utilities in the base-system of FreeBSD that can be used to work through a serial connection: cu(1) and tip(1).

For example, to connect from a client system that runs FreeBSD to the serial connection of another system:

```
# cu -l /dev/cuauN
```

Ports are numbered starting from zero. This means that COM1 is /dev/cuau0.

Additional programs are available through the Ports Collection, such as comms/minicom.

X Terminals

X terminals are the most sophisticated kind of terminal available. Instead of connecting to a serial port, they usually connect to a network like Ethernet. Instead of being relegated to text-only applications, they can display any Xorg application.

This chapter does not cover the setup, configuration, or use of X terminals.

30.3.1. Terminal Configuration

This section describes how to configure a FreeBSD system to enable a login session on a serial terminal. It assumes that the system recognizes the serial port to which the terminal is connected and that the terminal is connected with the correct cable.

In FreeBSD, `init` reads /etc/ttys and starts a `getty` process on the available terminals. The `getty` process is responsible for reading a login name and starting the `login` program. The ports on the FreeBSD system which allow logins are listed in /etc/ttys. For example, the first virtual console, ttyv0, has an entry in this file, allowing logins on the console. This file also contains entries for the other virtual consoles, serial ports, and pseudo-ttys. For a hardwired terminal, the serial port's /dev entry is listed without the `/dev` part. For example, /dev/ttyv0 is listed as `ttyv0`.

The default /etc/ttys configures support for the first four serial ports, ttyu0 through ttyu3:

```
ttyu0   "/usr/libexec/getty std.115200"   dialup  off secure
ttyu1   "/usr/libexec/getty std.115200"   dialup  off secure
ttyu2   "/usr/libexec/getty std.115200"   dialup  off secure
```

```
ttyu3   "/usr/libexec/getty std.115200"   dialup  off secure
```

When attaching a terminal to one of those ports, modify the default entry to set the required speed and terminal type, to turn the device `on` and, if needed, to change the port's `secure` setting. If the terminal is connected to another port, add an entry for the port.

Configuring Terminal Entries configures two terminals in /etc/ttys. The first entry configures a Wyse-50 connected to COM2. The second entry configures an old computer running Procomm terminal software emulating a VT-100 terminal. The computer is connected to the sixth serial port on a multi-port serial card.

Example 40. Configuring Terminal Entries

```
ttyu1  "/usr/libexec/getty std.38400"  wy50   on insecure
ttyu5   "/usr/libexec/getty std.19200"  vt100  on insecure
```

The first field specifies the device name of the serial terminal.

The second field tells `getty` to initialize and open the line, set the line speed, prompt for a user name, and then execute the `login` program. The optional *getty type* configures characteristics on the terminal line, like bps rate and parity. The available getty types are listed in /etc/gettytab. In almost all cases, the getty types that start with `std` will work for hardwired terminals as these entries ignore parity. There is a `std` entry for each bps rate from 110 to 115200. Refer to gettytab(5) for more information. When setting the getty type, make sure to match the communications settings used by the terminal. For this example, the Wyse-50 uses no parity and connects at 38400 bps. The computer uses no parity and connects at 19200 bps.

The third field is the type of terminal. For dial-up ports, `unknown` or `dialup` is typically used since users may dial up with practically any type of terminal or software. Since the terminal type does not change for hardwired terminals, a real terminal type from /etc/termcap can be specified. For this example, the Wyse-50 uses the real terminal type while the computer running Procomm is set to emulate a VT-100.

The fourth field specifies if the port should be enabled. To enable logins on this port, this field must be set to `on`.

The final field is used to specify whether the port is secure. Marking a port as `secure` means that it is trusted enough to allow `root` to login from that port. Insecure ports do not allow `root` logins. On an insecure port, users must login from unprivileged accounts and then use `su` or a similar mechanism to gain superuser privileges, as described in "The Superuser Account". For security reasons, it is recommended to change this setting to `insecure`.

After making any changes to /etc/ttys, send a SIGHUP (hangup) signal to the `init` process to force it to re-read its configuration file:

```
# kill -HUP 1
```

Since `init` is always the first process run on a system, it always has a process ID of `1`.

If everything is set up correctly, all cables are in place, and the terminals are powered up, a `getty` process should now be running on each terminal and login prompts should be available on each terminal.

30.3.2. Troubleshooting the Connection

Even with the most meticulous attention to detail, something could still go wrong while setting up a terminal. Here is a list of common symptoms and some suggested fixes.

If no login prompt appears, make sure the terminal is plugged in and powered up. If it is a personal computer acting as a terminal, make sure it is running terminal emulation software on the correct serial port.

Make sure the cable is connected firmly to both the terminal and the FreeBSD computer. Make sure it is the right kind of cable.

Make sure the terminal and FreeBSD agree on the bps rate and parity settings. For a video display terminal, make sure the contrast and brightness controls are turned up. If it is a printing terminal, make sure paper and ink are in good supply.

Use `ps` to make sure that a `getty` process is running and serving the terminal. For example, the following listing shows that a `getty` is running on the second serial port, ttyu1, and is using the `std.38400` entry in /etc/gettytab:

```
# ps -axww|grep ttyu
22189  d1  Is+    0:00.03 /usr/libexec/getty std.38400 ttyu1
```

If no `getty` process is running, make sure the port is enabled in /etc/ttys. Remember to run `kill -HUP 1` after modifying /etc/ttys.

If the `getty` process is running but the terminal still does not display a login prompt, or if it displays a prompt but will not accept typed input, the terminal or cable may not support hardware handshaking. Try changing the entry in /etc/ttys from `std.38400` to `3wire.38400`, then run `kill -HUP 1` after modifying /etc/ttys. The `3wire` entry is similar to `std`, but ignores hardware handshaking. The bps may also need to be reduced or software flow control enabled when using `3wire` to prevent buffer overflows.

If garbage appears instead of a login prompt, make sure the terminal and FreeBSD agree on the bps rate and parity settings. Check the `getty` processes to make sure the correct *getty* type is in use. If not, edit /etc/ttys and run `kill -HUP 1`.

If characters appear doubled and the password appears when typed, switch the terminal, or the terminal emulation software, from "half duplex" or "local echo" to "full duplex."

30.4. Dial-in Service

Configuring a FreeBSD system for dial-in service is similar to configuring terminals, except that modems are used instead of terminal devices. FreeBSD supports both external and internal modems.

External modems are more convenient because they often can be configured via parameters stored in non-volatile RAM and they usually provide lighted indicators that display the state of important RS-232 signals, indicating whether the modem is operating properly.

Internal modems usually lack non-volatile RAM, so their configuration may be limited to setting DIP switches. If the internal modem has any signal indicator lights, they are difficult to view when the system's cover is in place.

When using an external modem, a proper cable is needed. A standard RS-232C serial cable should suffice.

FreeBSD needs the RTS and CTS signals for flow control at speeds above 2400 bps, the CD signal to detect when a call has been answered or the line has been hung up, and the DTR signal to reset the modem after a session is complete. Some cables are wired without all of the needed signals, so if a login session does not go away when the line hangs up, there may be a problem with the cable. Refer to Serial Cables and Ports for more information about these signals.

Like other UNIX®-like operating systems, FreeBSD uses the hardware signals to find out when a call has been answered or a line has been hung up and to hangup and reset the modem after a call. FreeBSD avoids sending commands to the modem or watching for status reports from the modem.

FreeBSD supports the NS8250, NS16450, NS16550, and NS16550A-based RS-232C (CCITT V.24) communications interfaces. The 8250 and 16450 devices have single-character buffers. The 16550 device provides a 16-character buffer, which allows for better system performance. Bugs in plain 16550 devices prevent the use of the 16-character buffer, so use 16550A devices if possible. As single-character-buffer devices require more work by the operating system than the 16-character-buffer devices, 16550A-based serial interface cards are preferred. If the system has many active serial ports or will have a heavy load, 16550A-based cards are better for low-error-rate communications.

The rest of this section demonstrates how to configure a modem to receive incoming connections, how to communicate with the modem, and offers some troubleshooting tips.

30.4.1. Modem Configuration

As with terminals, `init` spawns a `getty` process for each configured serial port used for dial-in connections. When a user dials the modem's line and the modems connect, the "Carrier Detect" signal is reported by the modem. The kernel notices that the carrier has been detected and instructs `getty` to open the port and display a `login:` prompt at the specified initial line speed. In a typical configuration, if garbage characters are received, usually due to the modem's connection speed being different than the configured speed, `getty` tries adjusting the line speeds until it receives reasonable characters. After the user enters their login name, `getty` executes `login`, which completes the login process by asking for the user's password and then starting the user's shell.

There are two schools of thought regarding dial-up modems. One configuration method is to set the modems and systems so that no matter at what speed a remote user dials in, the dial-in RS-232 interface runs at a locked speed. The benefit of this configuration is that the remote user always sees a system login prompt immediately. The downside is that the system does not know what a user's true data rate is, so full-screen programs like Emacs will not adjust their screen-painting methods to make their response better for slower connections.

The second method is to configure the RS-232 interface to vary its speed based on the remote user's connection speed. As `getty` does not understand any particular modem's connection speed reporting, it gives a `login:` message at an initial speed and watches the characters that come back in response. If the user sees junk, they should press `Enter` until they see a recognizable prompt. If the data rates do not match, `getty` sees anything the user types as junk, tries the next speed, and gives the `login:` prompt again. This procedure normally only takes a keystroke or two before the user sees a good prompt. This login sequence does not look as clean as the locked-speed method, but a user on a low-speed connection should receive better interactive response from full-screen programs.

When locking a modem's data communications rate at a particular speed, no changes to /etc/gettytab should be needed. However, for a matching-speed configuration, additional entries may be required in order to define the speeds to use for the modem. This example configures a 14.4 Kbps modem with a top interface speed of 19.2 Kbps using 8-bit, no parity connections. It configures `getty` to start the communications rate for a V.32bis connection at 19.2 Kbps, then cycles through 9600 bps, 2400 bps, 1200 bps, 300 bps, and back to 19.2 Kbps. Communications rate cycling is implemented with the `nx=` (next table) capability. Each line uses a `tc=` (table continuation) entry to pick up the rest of the settings for a particular data rate.

```
#
# Additions for a V.32bis Modem
#
um|V300|High Speed Modem at 300,8-bit:\
        :nx=V19200:tc=std.300:
un|V1200|High Speed Modem at 1200,8-bit:\
        :nx=V300:tc=std.1200:
uo|V2400|High Speed Modem at 2400,8-bit:\
        :nx=V1200:tc=std.2400:
up|V9600|High Speed Modem at 9600,8-bit:\
        :nx=V2400:tc=std.9600:
uq|V19200|High Speed Modem at 19200,8-bit:\
        :nx=V9600:tc=std.19200:
```

For a 28.8 Kbps modem, or to take advantage of compression on a 14.4 Kbps modem, use a higher communications rate, as seen in this example:

```
#
# Additions for a V.32bis or V.34 Modem
# Starting at 57.6 Kbps
#
vm|VH300|Very High Speed Modem at 300,8-bit:\
        :nx=VH57600:tc=std.300:
vn|VH1200|Very High Speed Modem at 1200,8-bit:\
        :nx=VH300:tc=std.1200:
vo|VH2400|Very High Speed Modem at 2400,8-bit:\
        :nx=VH1200:tc=std.2400:
vp|VH9600|Very High Speed Modem at 9600,8-bit:\
        :nx=VH2400:tc=std.9600:
vq|VH57600|Very High Speed Modem at 57600,8-bit:\
        :nx=VH9600:tc=std.57600:
```

For a slow CPU or a heavily loaded system without 16550A-based serial ports, this configuration may produce `uart` "silo" errors at 57.6 Kbps.

The configuration of /etc/ttys is similar to Configuring Terminal Entries, but a different argument is passed to `getty` and `dialup` is used for the terminal type. Replace *xxx* with the process `init` will run on the device:

```
ttyu0   "/usr/libexec/getty xxx"   dialup on
```

The `dialup` terminal type can be changed. For example, setting `vt102` as the default terminal type allows users to use VT102 emulation on their remote systems.

For a locked-speed configuration, specify the speed with a valid type listed in /etc/gettytab. This example is for a modem whose port speed is locked at 19.2 Kbps:

```
ttyu0   "/usr/libexec/getty std.19200"   dialup on
```

In a matching-speed configuration, the entry needs to reference the appropriate beginning "auto-baud" entry in /etc/gettytab. To continue the example for a matching-speed modem that starts at 19.2 Kbps, use this entry:

```
ttyu0   "/usr/libexec/getty V19200"   dialup on
```

After editing /etc/ttys, wait until the modem is properly configured and connected before signaling `init`:

```
# kill -HUP 1
```

High-speed modems, like V.32, V.32bis, and V.34 modems, use hardware (`RTS/CTS`) flow control. Use `stty` to set the hardware flow control flag for the modem port. This example sets the `crtscts` flag on COM2's dial-in and dial-out initialization devices:

```
# stty -f /dev/ttyu1.init crtscts
# stty -f /dev/cuau1.init crtscts
```

30.4.2. Troubleshooting

This section provides a few tips for troubleshooting a dial-up modem that will not connect to a FreeBSD system.

Hook up the modem to the FreeBSD system and boot the system. If the modem has status indication lights, watch to see whether the modem's DTR indicator lights when the `login:` prompt appears on the system's console. If it lights up, that should mean that FreeBSD has started a `getty` process on the appropriate communications port and is waiting for the modem to accept a call.

If the DTR indicator does not light, login to the FreeBSD system through the console and type `ps ax` to see if FreeBSD is running a `getty` process on the correct port:

```
114 ??  I      0:00.10 /usr/libexec/getty V19200 ttyu0
```

If the second column contains a `d0` instead of a `??` and the modem has not accepted a call yet, this means that `getty` has completed its open on the communications port. This could indicate a problem with the cabling or a misconfigured modem because `getty` should not be able to open the communications port until the carrier detect signal has been asserted by the modem.

If no `getty` processes are waiting to open the port, double-check that the entry for the port is correct in /etc/ttys. Also, check /var/log/messages to see if there are any log messages from `init` or `getty`.

Next, try dialing into the system. Be sure to use 8 bits, no parity, and 1 stop bit on the remote system. If a prompt does not appear right away, or the prompt shows garbage, try pressing `Enter` about once per second. If there is still no `login:` prompt, try sending a `BREAK`. When using a high-speed modem, try dialing again after locking the dialing modem's interface speed.

If there is still no `login:` prompt, check /etc/gettytab again and double-check that:

- The initial capability name specified in the entry in /etc/ttys matches the name of a capability in /etc/gettytab.
- Each `nx=` entry matches another gettytab capability name.
- Each `tc=` entry matches another gettytab capability name.

If the modem on the FreeBSD system will not answer, make sure that the modem is configured to answer the phone when DTR is asserted. If the modem seems to be configured correctly, verify that the DTR line is asserted by checking the modem's indicator lights.

If it still does not work, try sending an email to the FreeBSD general questions mailing list describing the modem and the problem.

30.5. Dial-out Service

The following are tips for getting the host to connect over the modem to another computer. This is appropriate for establishing a terminal session with a remote host.

This kind of connection can be helpful to get a file on the Internet if there are problems using PPP. If

PPP is not working, use the terminal session to FTP the needed file. Then use zmodem to transfer it to the machine.

30.5.1. Using a Stock Hayes Modem

A generic Hayes dialer is built into `tip`. Use `at=hayes` in /etc/remote.

The Hayes driver is not smart enough to recognize some of the advanced features of newer modems messages like `BUSY`, `NO DIALTONE`, or `CONNECT 115200`. Turn those messages off when using `tip` with `ATX0&W`.

The dial timeout for `tip` is 60 seconds. The modem should use something less, or else `tip` will think there is a communication problem. Try `ATS7=45&W`.

30.5.2. Using `AT` Commands

Create a "direct" entry in /etc/remote. For example, if the modem is hooked up to the first serial port, /dev/cuau0, use the following line:

```
cuau0:dv=/dev/cuau0:br#19200:pa=none
```

Use the highest bps rate the modem supports in the `br` capability. Then, type `tip cuau0` to connect to the modem.

Or, use `cu` as `root` with the following command:

```
# cu -lline -sspeed
```

line is the serial port, such as /dev/cuau0, and *speed* is the speed, such as `57600`. When finished entering the AT commands, type `~.` to exit.

30.5.3. The `@` Sign Does Not Work

The `@` sign in the phone number capability tells `tip` to look in /etc/phones for a phone number. But, the `@` sign is also a special character in capability files like /etc/remote, so it needs to be escaped with a backslash:

```
pn=\@
```

30.5.4. Dialing from the Command Line

Put a "generic" entry in /etc/remote. For example:

```
tip115200|Dial any phone number at 115200 bps:\
        :dv=/dev/cuau0:br#115200:at=hayes:pa=none:du:
tip57600|Dial any phone number at 57600 bps:\
        :dv=/dev/cuau0:br#57600:at=hayes:pa=none:du:
```

This should now work:

```
# tip -115200 5551234
```

Users who prefer `cu` over `tip`, can use a generic `cu` entry:

```
cu115200|Use cu to dial any number at 115200bps:\
        :dv=/dev/cuau1:br#57600:at=hayes:pa=none:du:
```

and type:

```
# cu 5551234 -s 115200
```

30.5.5. Setting the bps Rate

Put in an entry for `tip1200` or `cu1200`, but go ahead and use whatever bps rate is appropriate with the `br` capability. `tip` thinks a good default is 1200 bps which is why it looks for a `tip1200` entry. 1200 bps does not have to be used, though.

30.5.6. Accessing a Number of Hosts Through a Terminal Server

Rather than waiting until connected and typing `CONNECT host` each time, use `tip`'s `cm` capability. For example, these entries in /etc/remote allow typing `tip pain` or `tip muffin` to connect to the hosts `pain` or `muffin`, and `tip deep13` to connect to the terminal server.

```
pain|pain.deep13.com|Forrester's machine:\
        :cm=CONNECT pain\n:tc=deep13:
muffin|muffin.deep13.com|Frank's machine:\
        :cm=CONNECT muffin\n:tc=deep13:
deep13:Gizmonics Institute terminal server:\
        :dv=/dev/cuau2:br#38400:at=hayes:du:pa=none:pn=5551234:
```

30.5.7. Using More Than One Line with `tip`

This is often a problem where a university has several modem lines and several thousand students trying to use them.

Make an entry in /etc/remote and use `@` for the `pn` capability:

```
big-university:\
        :pn=\@:tc=dialout
dialout:\
        :dv=/dev/cuau3:br#9600:at=courier:du:pa=none:
```

Then, list the phone numbers in /etc/phones:

```
big-university 5551111
big-university 5551112
big-university 5551113
big-university 5551114
```

`tip` will try each number in the listed order, then give up. To keep retrying, run `tip` in a `while` loop.

30.5.8. Using the Force Character

`Ctrl` + `P` is the default "force" character, used to tell `tip` that the next character is literal data. The force character can be set to any other character with the `~s` escape, which means "set a variable."

Type `~sforce=single-char` followed by a newline. *single-char* is any single character. If *single-char* is left out, then the force character is the null character, which is accessed by typing `Ctrl` + `2` or `Ctrl` + `Space`. A pretty good value for *single-char* is `Shift` + `Ctrl` + `6`, which is only used on some terminal servers.

To change the force character, specify the following in ~/.tiprc:

```
force=single-char
```

30.5.9. Upper Case Characters

This happens when `Ctrl` + `A` is pressed, which is `tip`'s "raise character", specially designed for people with broken caps-lock keys. Use `~s` to set `raisechar` to something reasonable. It can be set to be the same as the force character, if neither feature is used.

Here is a sample ~/.tiprc for Emacs users who need to type `Ctrl` + `2` and `Ctrl` + `A`:

```
force=^^
raisechar=^^
```

The `^^` is `Shift` + `Ctrl` + `6`.

30.5.10. File Transfers with `tip`

When talking to another UNIX®-like operating system, files can be sent and received using `~p` (put) and `~t` (take). These commands run `cat` and `echo` on the remote system to accept and send files. The syntax is: `~p` local-file [remote-file] `~t` remote-file [local-file]

There is no error checking, so another protocol, like zmodem, should probably be used.

30.5.11. Using zmodem with `tip`?

To receive files, start the sending program on the remote end. Then, type `~C rz` to begin receiving them locally.

To send files, start the receiving program on the remote end. Then, type `~C sz files` to send them to the remote system.

30.6. Setting Up the Serial Console

FreeBSD has the ability to boot a system with a dumb terminal on a serial port as a console. This configuration is useful for system administrators who wish to install FreeBSD on machines that have no keyboard or monitor attached, and developers who want to debug the kernel or device drivers.

As described in The FreeBSD Booting Process, FreeBSD employs a three stage bootstrap. The first two stages are in the boot block code which is stored at the beginning of the FreeBSD slice on the boot disk. The boot block then loads and runs the boot loader as the third stage code.

In order to set up booting from a serial console, the boot block code, the boot loader code, and the kernel need to be configured.

30.6.1. Quick Serial Console Configuration

This section provides a fast overview of setting up the serial console. This procedure can be used when the dumb terminal is connected to COM1.

Procedure: Configuring a Serial Console on COM1

1. Connect the serial cable to COM1 and the controlling terminal.
2. To configure boot messages to display on the serial console, issue the following command as the superuser:

```
# echo 'console="comconsole"' >> /boot/loader.conf
```

3. Edit /etc/ttys and change `off` to `on` and `dialup` to `vt100` for the ttyu0 entry. Otherwise, a password will not be required to connect via the serial console, resulting in a potential security hole.
4. Reboot the system to see if the changes took effect.

If a different configuration is required, see the next section for a more in-depth configuration explanation.

30.6.2. In-Depth Serial Console Configuration

This section provides a more detailed explanation of the steps needed to setup a serial console in FreeBSD.

Procedure: Configuring a Serial Console

1. Prepare a serial cable.

   Use either a null-modem cable or a standard serial cable and a null-modem adapter. See Serial Cables and Ports for a discussion on serial cables.

2. Unplug the keyboard.

   Many systems probe for the keyboard during the Power-On Self-Test (POST) and will generate an error if the keyboard is not detected. Some machines will refuse to boot until the keyboard is plugged in.

   If the computer complains about the error, but boots anyway, no further configuration is needed.

   If the computer refuses to boot without a keyboard attached, configure the BIOS so that it ignores this error. Consult the motherboard's manual for details on how to do this.

   > Try setting the keyboard to "Not installed" in the BIOS. This setting tells the BIOS not to probe for a keyboard at power-on so it should not complain if the keyboard is absent. If that option is not present in the BIOS, look for an "Halt on Error" option instead. Setting this to "All but Keyboard" or to "No Errors" will have the same effect.

   If the system has a PS/2® mouse, unplug it as well. PS/2® mice share some hardware with the keyboard and leaving the mouse plugged in can fool the keyboard probe into thinking the keyboard is still there.

   > While most systems will boot without a keyboard, quite a few will not boot without a graphics adapter. Some systems can be configured to boot with no graphics adapter by changing the "graphics adapter" setting in the BIOS configuration to "Not installed". Other systems do not support this option and will refuse to boot if there is no display hardware in the system. With these machines, leave some kind of graphics card plugged in, even if it is just a junky mono board. A monitor does not need to be attached.

3. Plug a dumb terminal, an old computer with a modem program, or the serial port on another UNIX® box into the serial port.
4. Add the appropriate `hint.uart.*` entries to /boot/device.hints for the serial port. Some multi-port cards also require kernel configuration options. Refer to uart(4) for the required options and device hints for each supported serial port.
5. Create boot.config in the root directory of the `a` partition on the boot drive.

   This file instructs the boot block code how to boot the system. In order to activate the serial console, one or more of the following options are needed. When using multiple options, include them all on the same line:

   `-h`

   Toggles between the internal and serial consoles. Use this to switch console devices. For instance, to boot from the internal (video) console, use `-h` to direct the boot loader and the kernel to use the serial port as its console device. Alternatively, to boot from the serial port, use `-h` to tell the boot loader and the kernel to use the video display as the console instead.

`-D`

Toggles between the single and dual console configurations. In the single configuration, the console will be either the internal console (video display) or the serial port, depending on the state of `-h`. In the dual console configuration, both the video display and the serial port will become the console at the same time, regardless of the state of `-h`. However, the dual console configuration takes effect only while the boot block is running. Once the boot loader gets control, the console specified by `-h` becomes the only console.

`-P`

Makes the boot block probe the keyboard. If no keyboard is found, the `-D` and `-h` options are automatically set.

> Due to space constraints in the current version of the boot blocks, `-P` is capable of detecting extended keyboards only. Keyboards with less than 101 keys and without F11 and F12 keys may not be detected. Keyboards on some laptops may not be properly found because of this limitation. If this is the case, do not use `-P`.

Use either `-P` to select the console automatically or `-h` to activate the serial console. Refer to boot(8) and boot.config(5) for more details.

The options, except for `-P`, are passed to the boot loader. The boot loader will determine whether the internal video or the serial port should become the console by examining the state of `-h`. This means that if `-D` is specified but `-h` is not specified in /boot.config, the serial port can be used as the console only during the boot block as the boot loader will use the internal video display as the console.

6. Boot the machine.

When FreeBSD starts, the boot blocks echo the contents of /boot.config to the console. For example:

```
/boot.config: -P
Keyboard: no
```

The second line appears only if `-P` is in /boot.config and indicates the presence or absence of the keyboard. These messages go to either the serial or internal console, or both, depending on the option in /boot.config:

Options	**Message goes to**
none	internal console
`-h`	serial console

Options	Message goes to
`-D`	serial and internal consoles
`-Dh`	serial and internal consoles
`-P`, keyboard present	internal console
`-P`, keyboard absent	serial console

After the message, there will be a small pause before the boot blocks continue loading the boot loader and before any further messages are printed to the console. Under normal circumstances, there is no need to interrupt the boot blocks, but one can do so in order to make sure things are set up correctly.

Press any key, other than `Enter`, at the console to interrupt the boot process. The boot blocks will then prompt for further action:

```
>> FreeBSD/i386 BOOT
Default: 0:ad(0,a)/boot/loader
boot:
```

Verify that the above message appears on either the serial or internal console, or both, according to the options in /boot.config. If the message appears in the correct console, press `Enter` to continue the boot process.

If there is no prompt on the serial terminal, something is wrong with the settings. Enter `-h` then `Enter` or `Return` to tell the boot block (and then the boot loader and the kernel) to choose the serial port for the console. Once the system is up, go back and check what went wrong.

During the third stage of the boot process, one can still switch between the internal console and the serial console by setting appropriate environment variables in the boot loader. See loader(8) for more information.

This line in /boot/loader.conf or /boot/loader.conf.local configures the boot loader and the kernel to send their boot messages to the serial console, regardless of the options in /boot.config:

```
console="comconsole"
```

That line should be the first line of /boot/loader.conf so that boot messages are displayed on the serial console as early as possible.

If that line does not exist, or if it is set to `console="vidconsole"`, the boot loader and the kernel will use whichever console is indicated by `-h` in the boot block. See loader.conf(5) for more information.

At the moment, the boot loader has no option equivalent to `-P` in the boot block, and there is no provision to automatically select the internal console and the serial console based on the presence of the keyboard.

While it is not required, it is possible to provide a `login` prompt over the serial line. To configure this, edit the entry for the serial port in /etc/ttys using the instructions in Terminal Configuration. If the speed of the serial port has been changed, change `std.115200` to match the new setting.

30.6.3. Setting a Faster Serial Port Speed

By default, the serial port settings are 115200 baud, 8 bits, no parity, and 1 stop bit. To change the default console speed, use one of the following options:

- Edit /etc/make.conf and set `BOOT_COMCONSOLE_SPEED` to the new console speed. Then, recompile and install the boot blocks and the boot loader:

  ```
  # cd /sys/boot
  # make clean
  # make
  # make install
  ```

  If the serial console is configured in some other way than by booting with `-h`, or if the serial console used by the kernel is different from the one used by the boot blocks, add the following option, with the desired speed, to a custom kernel configuration file and compile a new kernel:

  ```
  options CONSPEED=19200
  ```

- Add the `-S19200` boot option to /boot.config, replacing `19200` with the speed to use.
- Add the following options to /boot/loader.conf. Replace `115200` with the speed to use.

  ```
  boot_multicons="YES"
  boot_serial="YES"
  comconsole_speed="115200"
  console="comconsole,vidconsole"
  ```

30.6.4. Entering the DDB Debugger from the Serial Line

To configure the ability to drop into the kernel debugger from the serial console, add the following options to a custom kernel configuration file and compile the kernel using the instructions in Configuring the FreeBSD Kernel. Note that while this is useful for remote diagnostics, it is also dangerous if a spurious BREAK is generated on the serial port. Refer to ddb(4) and ddb(8) for more information about the kernel debugger.

```
options BREAK_TO_DEBUGGER
options DDB
```

Chapter 31. PPP

31.1. Synopsis

FreeBSD supports the Point-to-Point (PPP) protocol which can be used to establish a network or Internet connection using a dial-up modem. This chapter describes how to configure modem-based communication services in FreeBSD.

Read this chapter to learn:

- How to configure, use, and troubleshoot a PPP connection.
- How to set up PPP over Ethernet (PPPoE).
- How to set up PPP over ATM (PPPoA).

Before reading this chapter:

- Be familiar with basic network terminology.
- Understand the basics and purpose of a dial-up connection and PPP.

31.2. Configuring PPP

FreeBSD provides built-in support for managing dial-up PPP connections using ppp(8). The default FreeBSD kernel provides support for tun which is used to interact with a modem hardware. Configuration is performed by editing at least one configuration file, and configuration files containing examples are provided. Finally, `ppp` is used to start and manage connections.

In order to use a PPP connection, the following items are needed:

- A dial-up account with an Internet Service Provider (ISP).
- A dial-up modem.
- The dial-up number for the ISP.
- The login name and password assigned by the ISP.
- The IP address of one or more DNS servers. Normally, the ISP provides these addresses. If it did not, FreeBSD can be configured to use DNS negotiation.

If any of the required information is missing, contact the ISP.

The following information may be supplied by the ISP, but is not necessary:

- The IP address of the default gateway. If this information is unknown, the ISP will automatically provide the correct value during connection setup. When configuring PPP on FreeBSD, this address is referred to as `HISADDR`.
- The subnet mask. If the ISP has not provided one, `255.255.255.255` will be used in the ppp(8) configuration file. *

If the ISP has assigned a static IP address and hostname, it should be input into the configuration file. Otherwise, this information will be automatically provided during connection setup.

The rest of this section demonstrates how to configure FreeBSD for common PPP connection scenarios. The required configuration file is /etc/ppp/ppp.conf and additional files and examples are available in /usr/share/examples/ppp/.

> Throughout this section, many of the file examples display line numbers. These line numbers have been added to make it easier to follow the discussion and are not meant to be placed in the actual file.
>
> When editing a configuration file, proper indentation is important. Lines that end in a `:` start in the first column (beginning of the line) while all other lines should be indented as shown using spaces or tabs.

31.2.1. Basic Configuration

In order to configure a PPP connection, first edit /etc/ppp/ppp.conf with the dial-in information for the ISP. This file is described as follows:

```
1     default:
2       set log Phase Chat LCP IPCP CCP tun command
3       ident user-ppp VERSION
4       set device /dev/cuau0
5       set speed 115200
6       set dial "ABORT BUSY ABORT NO\\sCARRIER TIMEOUT 5 \
7                 \"\" AT OK-AT-OK ATE1Q0 OK \\dATDT\\T TIMEOUT 40 CONNECT"
8       set timeout 180
9       enable dns
10
11    provider:
12      set phone "(123) 456 7890"
13      set authname foo
14      set authkey bar
15      set timeout 300
16      set ifaddr x.x.x.x/0 y.y.y.y/0 255.255.255.255 0.0.0.0
17      add default HISADDR
```

Line 1

Identifies the `default` entry. Commands in this entry (lines 2 through 9) are executed automatically when `ppp` is run.

Line 2

Enables verbose logging parameters for testing the connection. Once the configuration is working satisfactorily, this line should be reduced to:

```
set log phase tun
```

Line 3

Displays the version of ppp(8) to the PPP software running on the other side of the connection.

Line 4

Identifies the device to which the modem is connected, where COM1 is /dev/cuau0 and COM2 is /dev/cuau1.

Line 5

Sets the connection speed. If `115200` does not work on an older modem, try `38400` instead.

Lines 6 & 7

The dial string written as an expect-send syntax. Refer to chat(8) for more information.

Note that this command continues onto the next line for readability. Any command in ppp.conf may do this if the last character on the line is `\`.

Line 8

Sets the idle timeout for the link in seconds.

Line 9

Instructs the peer to confirm the DNS settings. If the local network is running its own DNS server, this line should be commented out, by adding a `#` at the beginning of the line, or removed.

Line 10

A blank line for readability. Blank lines are ignored by ppp(8).

Line 11

Identifies an entry called `provider`. This could be changed to the name of the ISP so that `load ISP` can be used to start the connection.

Line 12

Use the phone number for the ISP. Multiple phone numbers may be specified using the colon (`:`) or pipe character (`|`) as a separator. To rotate through the numbers, use a colon. To always attempt to dial the first number first and only use the other numbers if the first number fails, use the pipe character. Always enclose the entire set of phone numbers between quotation marks (`"`) to prevent dialing failures.

Lines 13 & 14

Use the user name and password for the ISP.

Line 15

Sets the default idle timeout in seconds for the connection. In this example, the connection will be closed automatically after 300 seconds of inactivity. To prevent a timeout, set this value to zero.

Line 16

Sets the interface addresses. The values used depend upon whether a static IP address has been obtained from the ISP or if it instead negotiates a dynamic IP address during connection.

If the ISP has allocated a static IP address and default gateway, replace *x.x.x.x* with the static IP address and replace *y.y.y.y* with the IP address of the default gateway. If the ISP has only provided a static IP address without a gateway address, replace *y.y.y.y* with `10.0.0.2/0`.

If the IP address changes whenever a connection is made, change this line to the following value. This tells ppp(8) to use the IP Configuration Protocol (IPCP) to negotiate a dynamic IP address:

```
set ifaddr 10.0.0.1/0 10.0.0.2/0 255.255.255.255 0.0.0.0
```

Line 17

Keep this line as-is as it adds a default route to the gateway. The `HISADDR` will automatically be replaced with the gateway address specified on line 16. It is important that this line appears after line 16.

Depending upon whether ppp(8) is started manually or automatically, a /etc/ppp/ppp.linkup may also need to be created which contains the following lines. This file is required when running `ppp` in `-auto` mode. This file is used after the connection has been established. At this point, the IP address will have been assigned and it is now possible to add the routing table entries. When creating this file, make sure that *provider* matches the value demonstrated in line 11 of ppp.conf.

```
provider:
      add default HISADDR
```

This file is also needed when the default gateway address is "guessed" in a static IP address configuration. In this case, remove line 17 from ppp.conf and create /etc/ppp/ppp.linkup with the above two lines. More examples for this file can be found in /usr/share/examples/ppp/.

By default, `ppp` must be run as `root`. To change this default, add the account of the user who should run `ppp` to the `network` group in /etc/group.

Then, give the user access to one or more entries in /etc/ppp/ppp.conf with `allow`. For example, to give `fred` and `mary` permission to only the `provider:` entry, add this line to the `provider:` section:

```
allow users fred mary
```

To give the specified users access to all entries, put that line in the `default` section instead.

31.2.2. Advanced Configuration

It is possible to configure PPP to supply DNS and NetBIOS nameserver addresses on demand.

To enable these extensions with PPP version 1.x, the following lines might be added to the relevant section of /etc/ppp/ppp.conf.

```
enable msext
set ns 203.14.100.1 203.14.100.2
set nbns 203.14.100.5
```

And for PPP version 2 and above:

```
accept dns
set dns 203.14.100.1 203.14.100.2
set nbns 203.14.100.5
```

This will tell the clients the primary and secondary name server addresses, and a NetBIOS nameserver host.

In version 2 and above, if the `set dns` line is omitted, PPP will use the values found in /etc/resolv.conf.

PAP and CHAP Authentication

Some ISPs set their system up so that the authentication part of the connection is done using either of the PAP or CHAP authentication mechanisms. If this is the case, the ISP will not give a `login:` prompt at connection, but will start talking PPP immediately.

PAP is less secure than CHAP, but security is not normally an issue here as passwords, although being sent as plain text with PAP, are being transmitted down a serial line only. There is not much room for crackers to "eavesdrop".

The following alterations must be made:

```
13      set authname MyUserName
14      set authkey MyPassword
15      set login
```

Line 13

This line specifies the PAP/CHAP user name.Insert the correct value for *MyUserName*.

Line 14

This line specifies the PAP/CHAP password. Insert the correct value for *MyPassword*. An additional line may be added, such as:

```
16      accept PAP
```

or

```
16      accept CHAP
```

to make it obvious that this is the intention, but PAP and CHAP are both accepted by default.

Line 15

The ISP will not normally require a login to the server when using PAP or CHAP. Therefore, disable the "set login" string.

Using PPP Network Address Translation Capability

PPP has ability to use internal NAT without kernel diverting capabilities. This functionality may be enabled by the following line in /etc/ppp/ppp.conf:

```
nat enable yes
```

Alternatively, NAT may be enabled by command-line option `-nat`. There is also /etc/rc.conf knob named `ppp_nat`, which is enabled by default.

When using this feature, it may be useful to include the following /etc/ppp/ppp.conf options to enable incoming connections forwarding:

```
nat port tcp 10.0.0.2:ftp ftp
nat port tcp 10.0.0.2:http http
```

or do not trust the outside at all

```
nat deny_incoming yes
```

31.2.3. Final System Configuration

While `ppp` is now configured, some edits still need to be made to /etc/rc.conf.

Working from the top down in this file, make sure the `hostname=` line is set:

```
hostname="foo.example.com"
```

If the ISP has supplied a static IP address and name, use this name as the host name.

Look for the `network_interfaces` variable. To configure the system to dial the ISP on demand, make sure the tun0 device is added to the list, otherwise remove it.

```
network_interfaces="lo0 tun0"
ifconfig_tun0=
```

The `ifconfig_tun0` variable should be empty, and a file called /etc/start_if.tun0 should be created. This file should contain the line:

```
ppp -auto mysystem
```

This script is executed at network configuration time, starting the ppp daemon in automatic mode. If this machine acts as a gateway, consider including `-alias`. Refer to the manual page for further details.

Make sure that the router program is set to `NO` with the following line in /etc/rc.conf:

```
router_enable="NO"
```

It is important that the `routed` daemon is not started, as `routed` tends to delete the default routing table entries created by `ppp`.

It is probably a good idea to ensure that the `sendmail_flags` line does not include the `-q` option, otherwise `sendmail` will attempt to do a network lookup every now and then, possibly causing the machine to dial out. Try this:

```
sendmail_flags="-bd"
```

The downside is that `sendmail` is forced to re-examine the mail queue whenever the ppp link. To automate this, include `!bg` in ppp.linkup:

```
provider:
  delete ALL
  add 0 0 HISADDR
  !bg sendmail -bd -q30m
```

An alternative is to set up a "dfilter" to block SMTP traffic. Refer to the sample files for further details.

31.2.4. Using `ppp`

All that is left is to reboot the machine. After rebooting, either type:

```
# ppp
```

and then `dial provider` to start the PPP session, or, to configure `ppp` to establish sessions automatically when there is outbound traffic and start_if.tun0 does not exist, type:

```
# ppp -auto provider
```

It is possible to talk to the `ppp` program while it is running in the background, but only if a suitable diagnostic port has been set up. To do this, add the following line to the configuration:

```
set server /var/run/ppp-tun%d DiagnosticPassword 0177
```

This will tell PPP to listen to the specified UNIX® domain socket, asking clients for the specified password before allowing access. The `%d` in the name is replaced with the tun device number that is in use.

Once a socket has been set up, the pppctl(8) program may be used in scripts that wish to manipulate the running program.

31.2.5. Configuring Dial-in Services

"Dial-in Service" provides a good description on enabling dial-up services using getty(8).

An alternative to `getty` is comms/mgetty+sendfax port), a smarter version of `getty` designed with dial-up lines in mind.

The advantages of using `mgetty` is that it actively *talks* to modems, meaning if port is turned off in /etc/ttys then the modem will not answer the phone.

Later versions of `mgetty` (from 0.99beta onwards) also support the automatic detection of PPP streams, allowing clients scriptless access to the server.

Refer to http://mgetty.greenie.net/doc/mgetty_toc.html for more information on `mgetty`.

By default the comms/mgetty+sendfax port comes with the `AUTO_PPP` option enabled allowing `mgetty` to detect the LCP phase of PPP connections and automatically spawn off a ppp shell. However, since the default login/password sequence does not occur it is necessary to authenticate users using either PAP or CHAP.

This section assumes the user has successfully compiled, and installed the comms/mgetty+sendfax port on his system.

Ensure that /usr/local/etc/mgetty+sendfax/login.config has the following:

```
/AutoPPP/ -     - /etc/ppp/ppp-pap-dialup
```

This tells `mgetty` to run ppp-pap-dialup for detected PPP connections.

Create an executable file called /etc/ppp/ppp-pap-dialup containing the following:

```
#!/bin/sh
exec /usr/sbin/ppp -direct pap$IDENT
```

For each dial-up line enabled in /etc/ttys, create a corresponding entry in /etc/ppp/ppp.conf. This will happily co-exist with the definitions we created above.

```
pap:
  enable pap
  set ifaddr 203.14.100.1 203.14.100.20-203.14.100.40
  enable proxy
```

Each user logging in with this method will need to have a username/password in /etc/ppp/ppp.secret, or alternatively add the following option to authenticate users via PAP from /etc/passwd.

```
enable passwdauth
```

To assign some users a static IP number, specify the number as the third argument in /etc/ppp/ppp.secret. See /usr/share/examples/ppp/ppp.secret.sample for examples.

31.3. Troubleshooting PPP Connections

This section covers a few issues which may arise when using PPP over a modem connection. Some ISPs present the `ssword` prompt while others present `password`. If the `ppp` script is not written accordingly, the login attempt will fail. The most common way to debug `ppp` connections is by connecting manually as described in this section.

31.3.1. Check the Device Nodes

When using a custom kernel, make sure to include the following line in the kernel configuration file:

```
device   uart
```

The uart device is already included in the `GENERIC` kernel, so no additional steps are necessary in this case. Just check the `dmesg` output for the modem device with:

```
# dmesg | grep uart
```

This should display some pertinent output about the uart devices. These are the COM ports we need. If the modem acts like a standard serial port, it should be listed on uart1, or COM2. If so, a kernel rebuild is not required. When matching up, if the modem is on uart1, the modem device would be /dev/cuau1.

31.3.2. Connecting Manually

Connecting to the Internet by manually controlling `ppp` is quick, easy, and a great way to debug a connection or just get information on how the ISP treats `ppp` client connections. Lets start PPP from the command line. Note that in all of our examples we will use *example* as the hostname of the machine running PPP. To start `ppp`:

```
# ppp
```

```
ppp ON example> set device /dev/cuau1
```

This second command sets the modem device to cuau1.

```
ppp ON example> set speed 115200
```

This sets the connection speed to 115,200 kbps.

```
ppp ON example> enable dns
```

This tells `ppp` to configure the resolver and add the nameserver lines to /etc/resolv.conf. If `ppp` cannot determine the hostname, it can manually be set later.

```
ppp ON example> term
```

This switches to "terminal" mode in order to manually control the modem.

```
deflink: Entering terminal mode on /dev/cuau1
type '~h' for help
```

```
at
OK
atdt123456789
```

Use `at` to initialize the modem, then use `atdt` and the number for the ISP to begin the dial in process.

```
CONNECT
```

Confirmation of the connection, if we are going to have any connection problems, unrelated to hardware, here is where we will attempt to resolve them.

```
ISP Login:myusername
```

At this prompt, return the prompt with the username that was provided by the ISP.

```
ISP Pass:mypassword
```

At this prompt, reply with the password that was provided by the ISP. Just like logging into FreeBSD, the password will not echo.

```
Shell or PPP:ppp
```

Depending on the ISP, this prompt might not appear. If it does, it is asking whether to use a shell on the provider or to start `ppp`. In this example, `ppp` was selected in order to establish an Internet connection.

```
Ppp ON example>
```

Notice that in this example the first `p` has been capitalized. This shows that we have successfully connected to the ISP.

```
PPp ON example>
```

We have successfully authenticated with our ISP and are waiting for the assigned IP address.

```
PPP ON example>
```

We have made an agreement on an IP address and successfully completed our connection.

```
PPP ON example>add default HISADDR
```

Here we add our default route, we need to do this before we can talk to the outside world as currently the only established connection is with the peer. If this fails due to existing routes, put a bang character `!` in front of the `add`. Alternatively, set this before making the actual connection and it will negotiate a new route accordingly.

If everything went good we should now have an active connection to the Internet, which could be thrown into the background using `CTRL` + `z`. If `PPP` returns to `ppp` the connection has been lost. This is good to know because it shows the connection status. Capital P's represent a connection to the ISP and lowercase p's show that the connection has been lost.

31.3.3. Debugging

If a connection cannot be established, turn hardware flow CTS/RTS to off using `set ctsrts off`. This is mainly the case when connected to some PPP-capable terminal servers, where PPP hangs when it tries to write data to the communication link, and waits for a Clear To Send (CTS) signal which may never come. When using this option, include `set accmap` as it may be required to defeat hardware dependent on passing certain characters from end to end, most of the time XON/XOFF. Refer to ppp(8) for more information on this option and how it is used.

An older modem may need `set parity even`. Parity is set at none be default, but is used for error checking with a large increase in traffic, on older modems.

PPP may not return to the command mode, which is usually a negotiation error where the ISP is waiting for negotiating to begin. At this point, using `~p` will force ppp to start sending the configuration information.

If a login prompt never appears, PAP or CHAP authentication is most likely required. To use PAP or CHAP, add the following options to PPP before going into terminal mode:

```
ppp ON example> set authname myusername
```

Where *myusername* should be replaced with the username that was assigned by the ISP.

```
ppp ON example> set authkey mypassword
```

Where *mypassword* should be replaced with the password that was assigned by the ISP.

If a connection is established, but cannot seem to find any domain name, try to ping(8) an IP address. If there is 100 percent (100%) packet loss, it is likely that a default route was not assigned. Double check that `add default HISADDR` was set during the connection. If a connection can be made to a remote IP address, it is possible that a resolver address has not been added to /etc/resolv.conf. This file should look like:

```
domain example.com
nameserver x.x.x.x
nameserver y.y.y.y
```

Where *x.x.x.x* and *y.y.y.y* should be replaced with the IP address of the ISP's DNS servers.

To configure syslog(3) to provide logging for the PPP connection, make sure this line exists in /etc/syslog.conf:

```
!ppp
*.*     /var/log/ppp.log
```

31.4. Using PPP over Ethernet (PPPoE)

This section describes how to set up PPP over Ethernet (PPPoE).

Here is an example of a working ppp.conf:

```
default:
  set log Phase tun command # add more detailed logging when needed
  set ifaddr 10.0.0.1/0 10.0.0.2/0

name_of_service_provider:
  set device PPPoE:xl1 # replace xl1 with the Ethernet device
  set authname THELOGINNAME
  set authkey THEPASSWORD
  set dial
  set login
  add default HISADDR
```

As `root`, run:

```
# ppp -ddial name_of_service_provider
```

Add the following to /etc/rc.conf:

```
ppp_enable="YES"
ppp_mode="ddial"
ppp_nat="YES"   # when needing to enable nat for the local network, otherwise NO
ppp_profile="name_of_service_provider"
```

31.4.1. Using a PPPoE Service Tag

Sometimes it will be necessary to use a service tag to establish the connection. Service tags are used to distinguish between different PPPoE servers attached to a given network.

Any required service tag information should be in the documentation provided by the ISP.

As a last resort, one could try installing the net/rr-pppoe package or port. Bear in mind however, this may de-program the modem and render it useless, so think twice before doing it. Simply install the program shipped with the modem. Then, access the **System** menu from the program. The name of the profile should be listed there. It is usually *ISP*.

The profile name (service tag) will be used in the PPPoE configuration entry in ppp.conf as the provider part for `set device`. Refer to ppp(8) for full details. It should look like this:

```
set device PPPoE:xl1:ISP
```

Do not forget to change *xl1* to the proper device for the Ethernet card.

Do not forget to change *ISP* to the profile.

For additional information, refer to Cheaper Broadband with FreeBSD on DSL by Renaud Waldura.

31.4.2. PPPoE with a 3Com® HomeConnect™ ADSL Modem Dual Link

This modem does not follow the PPPoE specification defined in RFC 2516.

In order to make FreeBSD capable of communicating with this device, a sysctl must be set. This can be done automatically at boot time by updating /etc/sysctl.conf:

```
net.graph.nonstandard_pppoe=1
```

or can be done immediately with the command:

```
# sysctl net.graph.nonstandard_pppoe=1
```

Unfortunately, because this is a system-wide setting, it is not possible to talk to a normal PPPoE client or server and a 3Com® HomeConnect™ ADSL Modem at the same time.

31.5. Using PPP over ATM (PPPoA)

The following describes how to set up PPP over ATM (PPPoA). PPPoA is a popular choice among European DSL providers.

31.5.1. Using mpd

The mpd application can be used to connect to a variety of services, in particular PPTP services. It can be installed using the net/mpd5 package or port. Many ADSL modems require that a PPTP tunnel is created between the modem and computer.

Once installed, configure mpd to suit the provider's settings. The port places a set of sample configuration files which are well documented in /usr/local/etc/mpd/. A complete guide to configure mpd is available in HTML format in /usr/ports/shared/doc/mpd/. Here is a sample configuration for connecting to an ADSL service with mpd. The configuration is spread over two files, first the mpd.conf:

> This example mpd.conf only works with mpd 4.x.

```
default:
    load adsl

adsl:
    new -i ng0 adsl adsl
    set bundle authname username ①
    set bundle password password ②
    set bundle disable multilink

    set link no pap acfcomp protocomp
    set link disable chap
    set link accept chap
    set link keep-alive 30 10

    set ipcp no vjcomp
    set ipcp ranges 0.0.0.0/0 0.0.0.0/0

    set iface route default
    set iface disable on-demand
    set iface enable proxy-arp
    set iface idle 0

    open
```

① The username used to authenticate with the ISP.

② The password used to authenticate with the ISP.

Information about the link, or links, to establish is found in mpd.links. An example mpd.links to accompany the above example is given beneath:

```
adsl:
    set link type pptp
    set pptp mode active
    set pptp enable originate outcall
    set pptp self 10.0.0.1 ①
    set pptp peer 10.0.0.138 ②
```

① The IP address of FreeBSD computer running mpd.

② The IP address of the ADSL modem. The Alcatel SpeedTouch™ Home defaults to `10.0.0.138`.

It is possible to initialize the connection easily by issuing the following command as `root`:

```
# mpd -b adsl
```

To view the status of the connection:

```
% ifconfig ng0
ng0: flags=88d1<UP,POINTOPOINT,RUNNING,NOARP,SIMPLEX,MULTICAST> mtu 1500
     inet 216.136.204.117 --> 204.152.186.171 netmask 0xffffffff
```

Using mpd is the recommended way to connect to an ADSL service with FreeBSD.

31.5.2. Using pptpclient

It is also possible to use FreeBSD to connect to other PPPoA services using net/pptpclient.

To use net/pptpclient to connect to a DSL service, install the port or package, then edit /etc/ppp/ppp.conf. An example section of ppp.conf is given below. For further information on ppp.conf options consult ppp(8).

```
adsl:
 set log phase chat lcp ipcp ccp tun command
 set timeout 0
 enable dns
 set authname username ①
 set authkey password ②
 set ifaddr 0 0
 add default HISADDR
```

① The username for the DSL provider.

② The password for the account.

Since the account's password is added to ppp.conf in plain text form, make sure nobody can read the contents of this file:

```
# chown root:wheel /etc/ppp/ppp.conf
# chmod 600 /etc/ppp/ppp.conf
```

This will open a tunnel for a PPP session to the DSL router. Ethernet DSL modems have a preconfigured LAN IP address to connect to. In the case of the Alcatel SpeedTouch™ Home, this address is `10.0.0.138`. The router's documentation should list the address the device uses. To open the tunnel and start a PPP session:

```
# pptp address adsl
```

If an ampersand ("&") is added to the end of this command, pptp will return the prompt.

A tun virtual tunnel device will be created for interaction between the pptp and ppp processes. Once the prompt is returned, or the pptp process has confirmed a connection, examine the tunnel:

```
% ifconfig tun0
tun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 1500
        inet 216.136.204.21 --> 204.152.186.171 netmask 0xffffff00
    Opened by PID 918
```

If the connection fails, check the configuration of the router, which is usually accessible using a web browser. Also, examine the output of `pptp` and the contents of the log file, /var/log/ppp.log for clues.

Chapter 32. Electronic Mail

32.1. Synopsis

"Electronic Mail", better known as email, is one of the most widely used forms of communication today. This chapter provides a basic introduction to running a mail server on FreeBSD, as well as an introduction to sending and receiving email using FreeBSD. For a complete coverage of this subject, refer to the books listed in Bibliography.

This chapter covers:

- Which software components are involved in sending and receiving electronic mail.
- How to configure DragonFly Mail Agent.
- Where basic Sendmail configuration files are located in FreeBSD.
- The difference between remote and local mailboxes.
- How to install and configure an alternate Mail Transfer Agent, replacing DragonFly Mail Agent or Sendmail.
- How to troubleshoot common mail server problems.
- How to configure Sendmail to only send mail.
- How to configure SMTP authentication for added security in Sendmail.
- How to install and use a Mail User Agent, such as mutt, to send and receive email.
- How to download mail from a remote POP or IMAP server.
- How to automatically apply filters and rules to incoming email.

32.2. Mail Components

There are five major parts involved in an email exchange: the Mail User Agent (MUA), the Mail Transfer Agent (MTA), a mail host, a remote or local mailbox, and DNS. This section provides an overview of these components.

Mail User Agent (MUA)

The Mail User Agent (MUA) is an application which is used to compose, send, and receive emails. This application can be a command line program, such as the built-in `mail` utility or a third-party application from the Ports Collection, such as alpine, elm, or mutt. Dozens of graphical programs are also available in the Ports Collection, including Claws Mail, Evolution, and Thunderbird. Some organizations provide a web mail program which can be accessed through a web browser. More information about installing and using a MUA on FreeBSD can be found in Mail User Agents.

Mail Transfer Agent (MTA)

The Mail Transfer Agent (MTA) is responsible for receiving incoming mail and delivering outgoing mail. Starting with FreeBSD version 14.0, the default MTA is DragonFly Mail Agent (dma(8)); in earlier versions, it is sendmail(8). Other MTAs, including Exim, Postfix, and qmail, may be installed to replace the default MTA.

Mail Host and Mailboxes

The mail host is a server that is responsible for delivering and receiving mail for a host or a network. The mail host collects all mail sent to the domain and stores it either in the default `mbox` or the alternative Maildir format, depending on the configuration. Once mail has been stored, it may either be read locally using a MUA or remotely accessed and collected using protocols such as POP or IMAP. If mail is read locally, a POP or IMAP server does not need to be installed.

Domain Name System (DNS)

The Domain Name System (DNS) and its daemon named(8) play a large role in the delivery of mail. In order to deliver mail from one site to another, the MTA will look up the remote site in DNS to determine which host will receive mail for the destination. This process also occurs when mail is sent from a remote host to the MTA.

32.3. DragonFly Mail Agent (DMA)

DragonFly Mail Agent (DMA) is the default MTA in FreeBSD starting with version 14.0. dma(8) is a small Mail Transport Agent (MTA), designed for home and office use. It accepts mails from locally installed Mail User Agents (MUA) and delivers the mails either locally or to a remote destination. Remote delivery includes several features like TLS/SSL support and SMTP authentication.

dma(8) is not intended as a replacement for real, big MTAs like sendmail(8) or postfix(1). Consequently, dma(8) does not listen on port 25 for incoming connections.

32.3.1. Configuring DragonFly Mail Agent (DMA)

DMA comes with a default configuration that will be suitable for many deployments. Custom settings are defined in /etc/dma/dma.conf, and SMTP authentication is configured in /etc/dma/auth.conf.

Using DMA to Route Outgoing Mail through Gmail (STARTTLS:SMTP example)

This example /etc/dma/dma.conf can be used to send mail using Google’s SMTP servers.

```
SMARTHOST smtp.gmail.com
PORT 587
AUTHPATH /etc/dma/auth.conf
SECURETRANSFER
STARTTLS
MASQUERADE username@gmail.com
```

Authentication can be set with one line in /etc/dma/auth.conf:

```
username@gmail.com|smtp.gmail.com:password
```

With enabled 2-factor authentication a generated application-specific password as the normal login password will be rejected. See Google documentation for more information on app-specific passwords.

Execute the following command to test the configuration:

```
% echo this is a test | mail -v -s testing-email username@gmail.com
```

Using DMA to Route Outgoing Mail through Fastmail (SSL/TLS example)

This example /etc/dma/dma.conf can be used to send mail using Fastmail's SMTP servers.

```
SMARTHOST smtp.fastmail.com
PORT 465
AUTHPATH /etc/dma/auth.conf
SECURETRANSFER
MAILNAME example.server.com
```

Authentication can be set with one line in /etc/dma/auth.conf:

```
username@fastmail.com|smtp.fastmail.com:password
```

Execute the following command to test the configuration:

```
% echo this is a test | mail -v -s testing-email username@fastmail.com
```

Using DMA to Route Outgoing Mail through a Custom Mail Host

This example /etc/dma/dma.conf can be used to send mail using a custom mail host.

```
SMARTHOST mail.example.org
PORT 587
AUTHPATH /etc/dma/auth.conf
SECURETRANSFER
STARTTLS
```

Authentication can be set with one line in /etc/dma/auth.conf:

```
username@example.org|mail.example.org:password
```

Execute the following command to test the configuration:

```
% echo this is a test | mail -v -s testing-email username@example.org
```

32.4. Sendmail

Sendmail is a venerable and versatile Mail Transfer Agent (MTA) with a long history in UNIX® and UNIX-like systems. It was a part of the FreeBSD base system until FreeBSD 13, offering robust email transport capabilities, extensive customization options, and support for complex routing and filtering.

32.4.1. Configuration Files

The configuration files for Sendmail are located in /etc/mail/.

/etc/mail/access

This access database file defines which hosts or IP addresses have access to the local mail server and what kind of access they have. Hosts listed as `OK`, which is the default option, are allowed to send mail to this host as long as the mail's final destination is the local machine. Hosts listed as `REJECT` are rejected for all mail connections. Hosts listed as `RELAY` are allowed to send mail for any destination using this mail server. Hosts listed as `ERROR` will have their mail returned with the specified mail error. If a host is listed as `SKIP`, Sendmail will abort the current search for this entry without accepting or rejecting the mail. Hosts listed as `QUARANTINE` will have their messages held and will receive the specified text as the reason for the hold.

Examples of using these options for both IPv4 and IPv6 addresses can be found in the FreeBSD sample configuration, /etc/mail/access.sample:

To configure the access database, use the format shown in the sample to make entries in /etc/mail/access, but do not put a comment symbol (`#`) in front of the entries. Create an entry for each host or network whose access should be configured. Mail senders that match the left side of the table are affected by the action on the right side of the table.

Whenever this file is updated, update its database and restart Sendmail:

```
# makemap hash /etc/mail/access < /etc/mail/access
# service sendmail restart
```

/etc/mail/aliases

This database file contains a list of virtual mailboxes that are expanded to users, files, programs, or other aliases. Here are a few entries to illustrate the file format:

```
root: localuser
ftp-bugs: joe,eric,paul
bit.bucket:  /dev/null
procmail: "|/usr/local/bin/procmail"
```

The mailbox name on the left side of the colon is expanded to the target(s) on the right. The first entry expands the `root` mailbox to the `localuser` mailbox, which is then looked up in the /etc/mail/aliases database. If no match is found, the message is delivered to `localuser`. The second entry shows a mail list. Mail to `ftp-bugs` is expanded to the three local mailboxes `joe`, `eric`, and `paul`. A remote mailbox could be specified as *user@example.com*. The third entry shows how to write mail to a file, in this case /dev/null. The last entry demonstrates how to send mail to a program, /usr/local/bin/procmail, through a UNIX® pipe. Refer to aliases(5) for more information about the format of this file.

Whenever this file is updated, run `newaliases` to update and initialize the aliases database.

/etc/mail/sendmail.cf

This is the master configuration file for Sendmail. It controls the overall behavior of Sendmail, including everything from rewriting email addresses to printing rejection messages to remote mail servers. Accordingly, this configuration file is quite complex. Fortunately, this file rarely needs to be changed for standard mail servers.

The master Sendmail configuration file can be built from m4(1) macros that define the features and behavior of Sendmail. Refer to /usr/src/contrib/sendmail/cf/README for some of the details.

Whenever changes to this file are made, Sendmail needs to be restarted for the changes to take effect.

/etc/mail/virtusertable

This database file maps mail addresses for virtual domains and users to real mailboxes. These mailboxes can be local, remote, aliases defined in /etc/mail/aliases, or files. This allows multiple virtual domains to be hosted on one machine.

FreeBSD provides a sample configuration file in /etc/mail/virtusertable.sample to further demonstrate its format. The following example demonstrates how to create custom entries using that format:

```
root@example.com                root
postmaster@example.com          postmaster@noc.example.net
@example.com                    joe
```

This file is processed in a first match order. When an email address matches the address on the left, it is mapped to the local mailbox listed on the right. The format of the first entry in this example maps a specific email address to a local mailbox, whereas the format of the second entry maps a specific email address to a remote mailbox. Finally, any email address from `example.com` which has not matched any of the previous entries will match the last mapping and be sent to the local mailbox `joe`. When creating custom entries, use this format and add them to /etc/mail/virtusertable. Whenever this file is edited, update its database and restart Sendmail:

```
# makemap hash /etc/mail/virtusertable < /etc/mail/virtusertable
# service sendmail restart
```

/etc/mail/relay-domains

In a default FreeBSD installation, Sendmail is configured to only send mail from the host it is running on. For example, if a POP server is available, users will be able to check mail from remote locations but they will not be able to send outgoing emails from outside locations. Typically, a few moments after the attempt, an email will be sent from `MAILER-DAEMON` with a `5.7 Relaying Denied` message.

The most straightforward solution is to add the ISP's FQDN to /etc/mail/relay-domains. If multiple addresses are needed, add them one per line:

```
your.isp.example.com
other.isp.example.net
users-isp.example.org
www.example.org
```

After creating or editing this file, restart Sendmail with `service sendmail restart`.

Now any mail sent through the system by any host in this list, provided the user has an account on the system, will succeed. This allows users to send mail from the system remotely without opening the system up to relaying SPAM from the Internet.

32.5. Changing the Mail Transfer Agent

Starting with FreeBSD version 14.0, dma(8) is the default MTA, and before 14.0, the default MTA is sendmail(8). However, the system administrator can change the system's MTA. A wide choice of alternative MTAs is available from the `mail` category of the FreeBSD Ports Collection.

> If the default's outgoing mail service is disabled, it is important that it is replaced with an alternative mail delivery system. Otherwise, system functions such as periodic(8) will be unable to deliver their results by email. Many parts of the system expect a functional MTA. If applications continue to use the default binaries to try to send email after they are disabled, mail could go into an inactive queue and never be delivered.

32.5.1. Replacing Sendmail with Other MTA

In order to completely disable sendmail(8) execute the following commands:

```
# sysrc sendmail_enable="NO"
# sysrc sendmail_submit_enable="NO"
# sysrc sendmail_outbound_enable="NO"
# sysrc sendmail_msp_queue_enable="NO"
```

To only disable sendmail(8)'s incoming mail service execute the following command:

```
# sysrc sendmail_enable="NO"
```

Then stop the sendmail(8) service:

```
# service sendmail onestop
```

Some extra configuration is needed as sendmail(8) is so ubiquitous that some software assumes it is already installed and configured. Check /etc/periodic.conf and make sure that these values are set to `NO`. If this file does not exist, create it with these entries:

```
daily_clean_hoststat_enable="NO"
daily_status_mail_rejects_enable="NO"
daily_status_include_submit_mailq="NO"
daily_submit_queuerun="NO"
```

The next step is to install another MTA, dma(8) will be used in this example. As pointed above, dma(8) is the default MTA in FreeBSD starting with version 14.0. Therefore, it is only necessary to install it from the ports when using a previous version.

To install it execute the following command:

```
# pkg install dma
```

Perform the configuration as indicated in Configuring DragonFly Mail Agent (DMA).

Then change all the entries in the file /etc/mail/mailer.conf to dma(8):

```
# Execute the "real" sendmail program, named /usr/libexec/sendmail/sendmail
#
# If dma(8) is installed, an example mailer.conf that uses dma(8) instead can
# be found in /usr/share/examples/dma
#
sendmail        /usr/local/libexec/dma
mailq           /usr/local/libexec/dma
newaliases      /usr/local/libexec/dma
```

When using the version of dma(8) included in the base system, the paths will change to /usr/libexec/dma.

To ensure that anything in the queue is flushed at boot or before shutdown, execute the following command:

```
# sysrc dma_flushq_enable="YES"
```

Once everything is configured, it is recommended to reboot the system. Rebooting provides the opportunity to ensure that the system is correctly configured to start the new MTA automatically on boot.

32.5.2. Replacing DragonFly Mail Agent (DMA) with Other MTA

As noted above, starting with FreeBSD version 14.0, the default MTA is DMA. In this example, mail/postfix will be used as the alternative MTA.

Before installing mail/postfix some extra configuration is needed. Check /etc/periodic.conf and make sure that these values are set to `NO`. If this file does not exist, create it with these entries:

```
daily_clean_hoststat_enable="NO"
daily_status_mail_rejects_enable="NO"
daily_status_include_submit_mailq="NO"
daily_submit_queuerun="NO"
```

Then install mail/postfix:

```
# pkg install postfix
```

To start mail/postfix at system boot execute the following command:

```
# sysrc postfix_enable="YES"
```

It is good practice to read the installation message after installing an application. Provides useful information about settings, etc.

If postfix is **not** already activated in /usr/local/etc/mail/mailer.conf execute the following commands:

```
mv /usr/local/etc/mail/mailer.conf /usr/local/etc/mail/mailer.conf.old
install -d /usr/local/etc/mail
install -m 0644 /usr/local/share/postfix/mailer.conf.postfix
/usr/local/etc/mail/mailer.conf
```

When employing SASL, ensure that postfix has access to read the sasldb file. This is accomplished by adding postfix to group mail and making the /usr/local/etc/sasldb* file(s) readable by group mail (this should be the default for new installs).

Once everything is configured, it is recommended to reboot the system. Rebooting provides the opportunity to ensure that the system is correctly configured to start the new MTA automatically on boot.

32.6. Mail User Agents

A MUA is an application that is used to send and receive email. As email "evolves" and becomes more complex, MUAs are becoming increasingly powerful and provide users increased functionality and flexibility. The `mail` category of the FreeBSD Ports Collection contains numerous MUAs. These include graphical email clients such as Evolution or Balsa and console based clients such as mutt or alpine.

32.6.1. mail

mail(1) is the default MUA installed with FreeBSD. It is a console based MUA that offers the basic functionality required to send and receive text-based email. It provides limited attachment support and can only access local mailboxes.

Although mail(1) does not natively support interaction with POP or IMAP servers, these mailboxes may be downloaded to a local `mbox` using an application such as fetchmail or getmail.

In order to send and receive email, run mail(1):

```
% mail
```

The contents of the user's mailbox in /var/mail are automatically read by mail(1). Should the mailbox be empty, the utility exits with a message indicating that no mail could be found. If mail exists, the application interface starts, and a list of messages will be displayed.

Messages are automatically numbered, as can be seen in the following example:

```
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/username": 3 messages 3 new
>N  1 root@localhost        Mon Mar  8 14:05  14/510   "test"
 N  2 root@localhost        Mon Mar  8 14:05  14/509   "user account"
 N  3 root@localhost        Mon Mar  8 14:05  14/509   "sample"
```

Messages can now be read by typing `t` followed by the message number.

This example reads the first email:

```
& t 1
Message 1:
From root@localhost  Mon Mar  8 14:05:52 2004
X-Original-To: username@localhost
Delivered-To: username@localhost
To: username@localhost
Subject: test
Date: Mon,  8 Mar 2004 14:05:52 +0200 (SAST)
From: root@localhost (Charlie Root)
```

```
This is a test message, please reply if you receive it.
```

As seen in this example, the message will be displayed with full headers.

To display the list of messages again, press `h`.

If the email requires a reply, press either `R` or `r` mail(1) keys. `R` instructs mail(1) to reply only to the sender of the email, while `r` replies to all other recipients of the message. These commands can be suffixed with the mail number of the message to reply to. After typing the response, the end of the message should be marked by a single `.` on its own line.

An example can be seen below:

```
& R 1
To: root@localhost
Subject: Re: test

Thank you, I did get your email.
.
EOT
```

In order to send a new email, press `m`, followed by the recipient email address. Multiple recipients may be specified by separating each address with the `,` delimiter. The subject of the message may then be entered, followed by the message contents. The end of the message should be specified by putting a single `.` on its own line.

```
& mail root@localhost
Subject: I mastered mail

Now I can send and receive email using mail ... :)
.
EOT
```

While using mail(1), press `?` to display help at any time. Refer to mail(1) for more help on how to use mail(1).

mail(1) was not designed to handle attachments and thus deals with them poorly. Newer MUAs handle attachments in a more intelligent way.

32.6.2. Mutt

Mutt is a powerful MUA, with many features, including:

- The ability to thread messages.
- PGP support for digital signing and encryption of email.
- MIME support.

- Maildir support.
- Highly customizable.

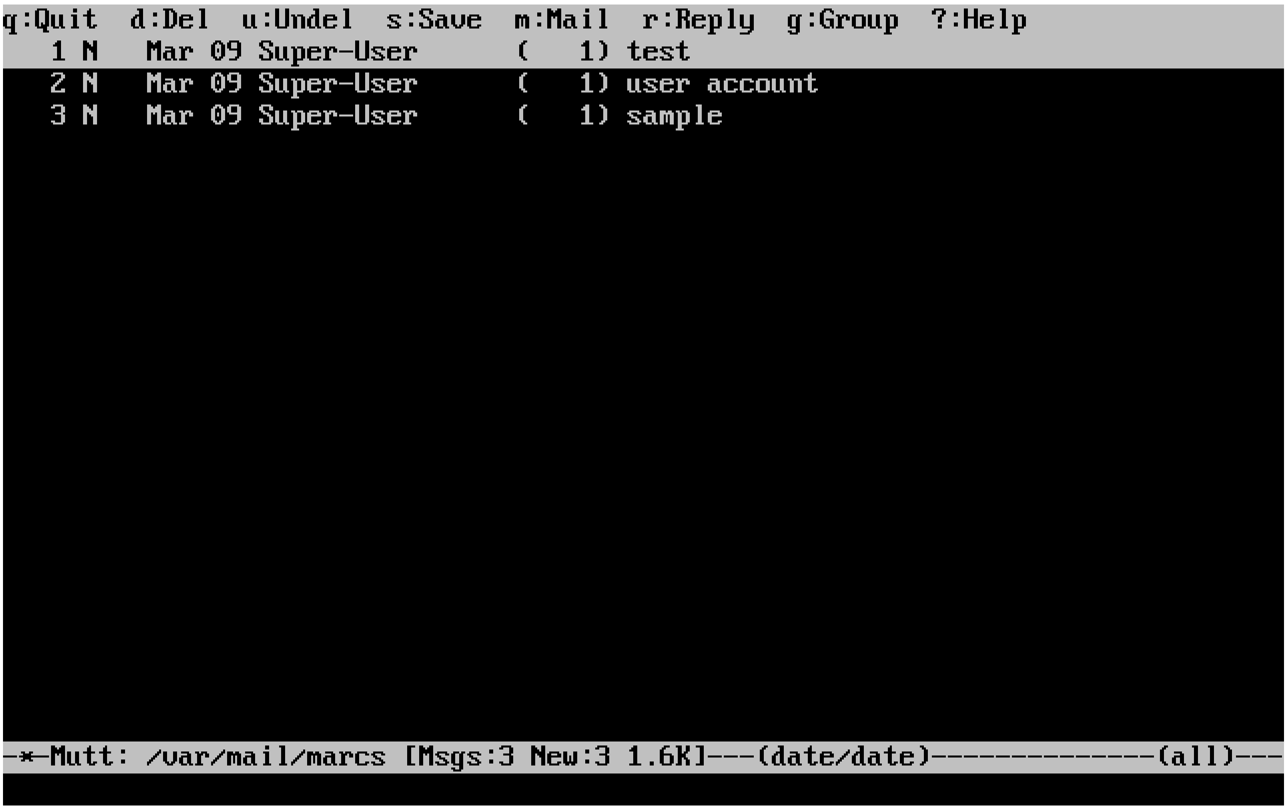
Refer to http://www.mutt.org for more information on Mutt.

A Mutt fork called NeoMutt is worth mentioning, which brings added features. See more on the NeoMutt website. If NeoMutt was chosen, replace the following command examples from `mutt` to `neomutt`.

Mutt may be installed using the mail/mutt port. After the port has been installed, Mutt can be started by issuing the following command:

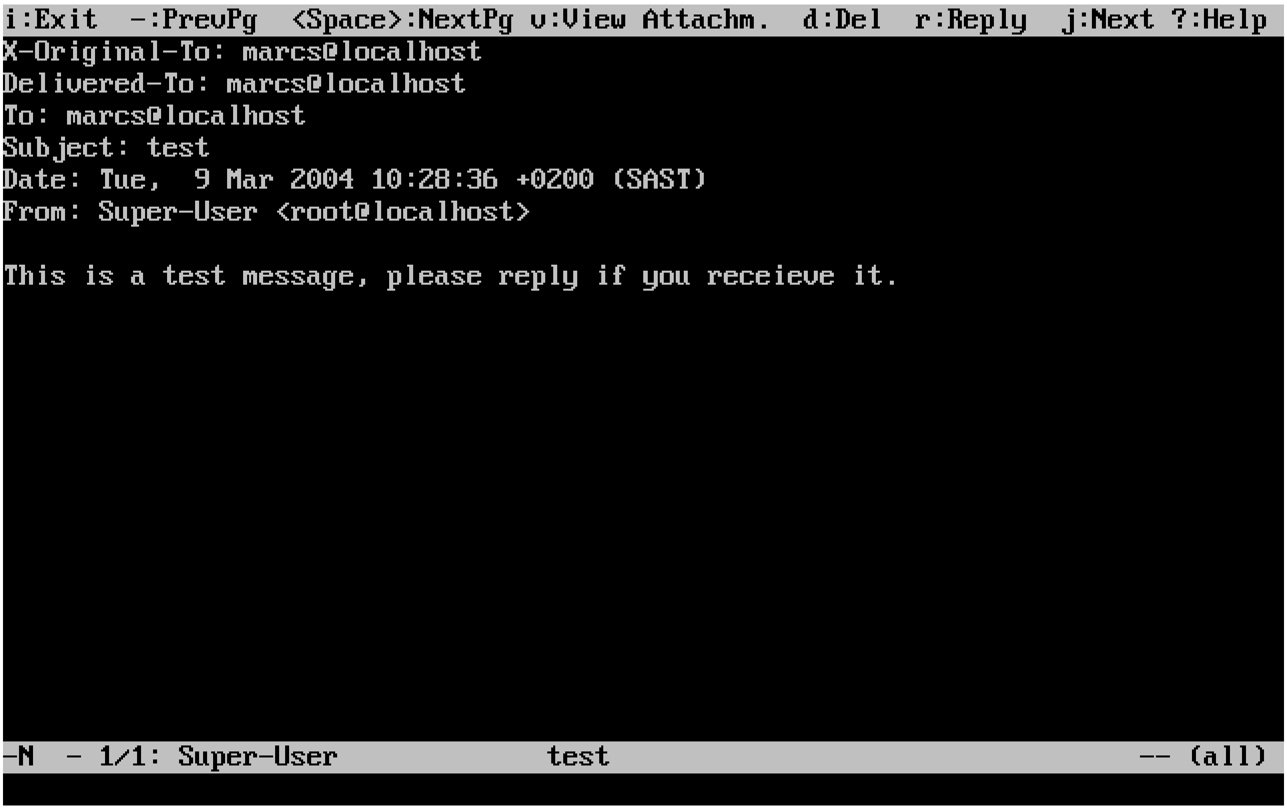
```
% mutt
```

Mutt will automatically read and display the contents of the user mailbox in /var/mail. If no mails are found, Mutt will wait for commands from the user. The example below shows Mutt displaying a list of messages:



```
q:Quit  d:Del  u:Undel  s:Save  m:Mail  r:Reply  g:Group  ?:Help
   1 N   Mar 09 Super-User       (   1) test
   2 N   Mar 09 Super-User       (   1) user account
   3 N   Mar 09 Super-User       (   1) sample

-*-Mutt: /var/mail/marcs [Msgs:3 New:3 1.6K]---(date/date)--------------(all)---
```

To read an email, select it using the cursor keys and press `Enter`. An example of Mutt displaying email can be seen below:

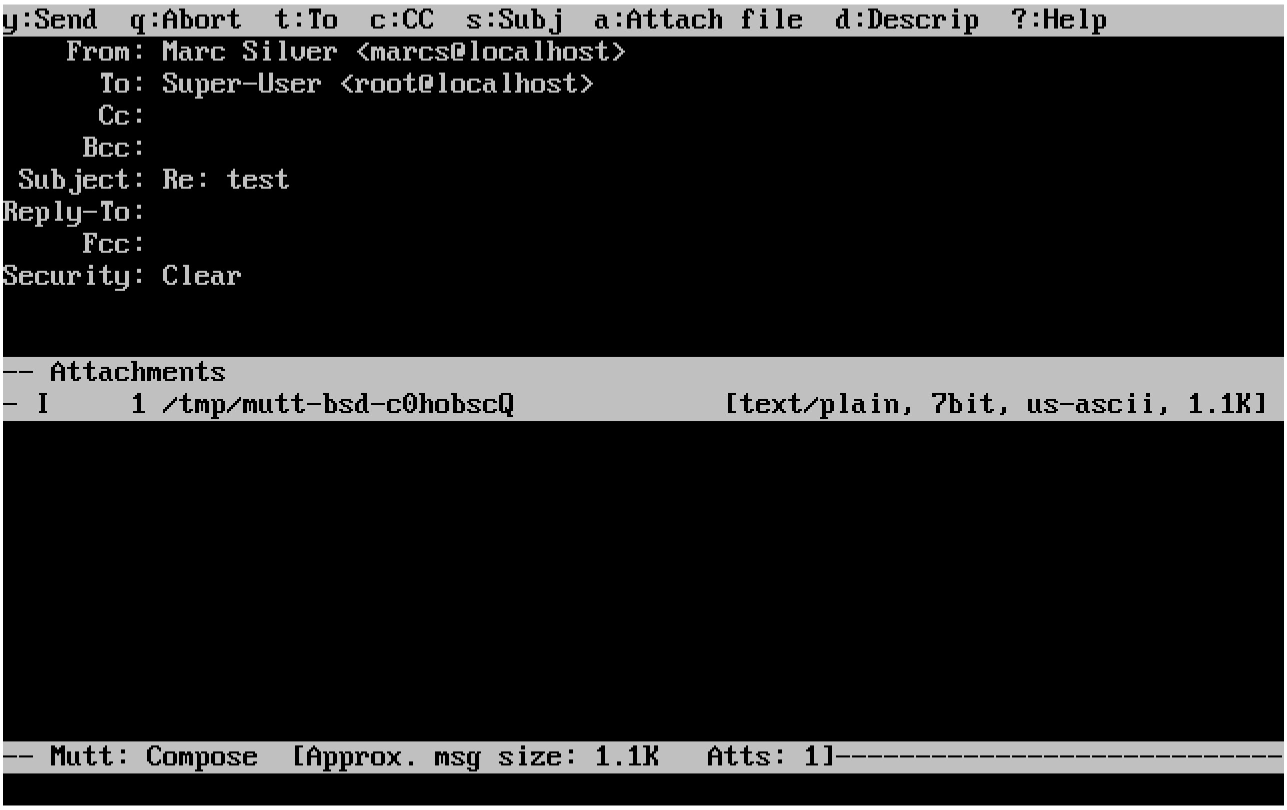


```
i:Exit  -:PrevPg  <Space>:NextPg v:View Attachm.  d:Del  r:Reply  j:Next ?:Help
X-Original-To: marcs@localhost
Delivered-To: marcs@localhost
To: marcs@localhost
Subject: test
Date: Tue,  9 Mar 2004 10:28:36 +0200 (SAST)
From: Super-User <root@localhost>

This is a test message, please reply if you receieve it.
















-N  - 1/1: Super-User              test                                -- (all)
```

Similar to mail(1), Mutt can be used to reply only to the sender of the message as well as to all recipients. To reply only to the sender of the email, press `r`. To send a group reply to the original sender as well as all the message recipients, press `g`.

By default, Mutt uses the vi(1) editor for creating and replying to emails. Each user can customize this by creating or editing the .muttrc in their home directory and setting the `editor` variable or by setting the `EDITOR` environment variable. Refer to http://www.mutt.org/ for more information about configuring Mutt.

To compose a new mail message, press `m`. After a valid subject has been given, Mutt will start vi(1) so the email can be written. Once the contents of the email are complete, save and quit from `vi`. Mutt will resume, displaying a summary screen of the mail that is to be delivered. In order to send the mail, press `y`. An example of the summary screen can be seen below:



```
y:Send  q:Abort  t:To  c:CC  s:Subj  a:Attach file  d:Descrip  ?:Help
    From: Marc Silver <marcs@localhost>
      To: Super-User <root@localhost>
      Cc:
     Bcc:
 Subject: Re: test
Reply-To:
     Fcc:
Security: Clear


-- Attachments
- I     1 /tmp/mutt-bsd-c0hobscQ             [text/plain, 7bit, us-ascii, 1.1K]









-- Mutt: Compose  [Approx. msg size: 1.1K   Atts: 1]----------------------------
```

Mutt contains extensive help which can be accessed from most of the menus by pressing `?`. The top line also displays the keyboard shortcuts where appropriate.

32.6.3. alpine

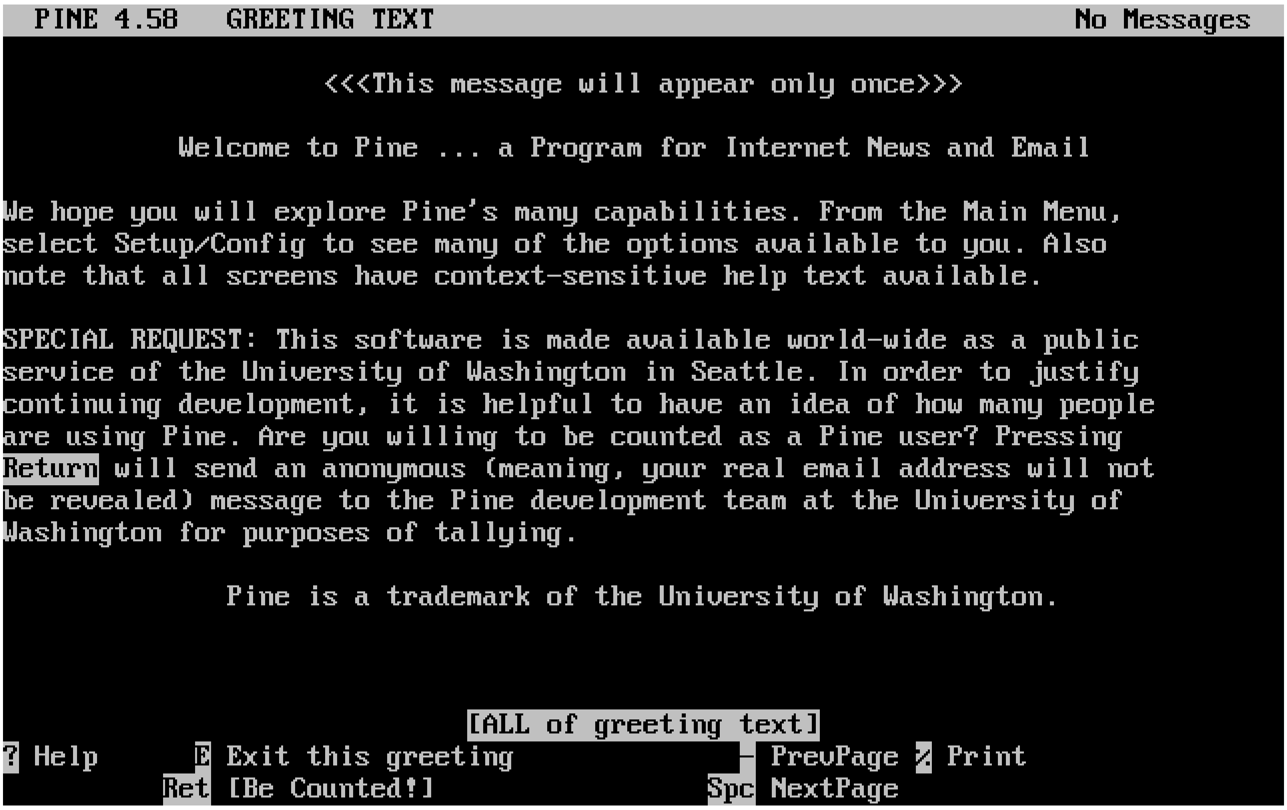
alpine is aimed at a beginner user, but also includes some advanced features.

> alpine has had several remote vulnerabilities discovered in the past, which allowed remote attackers to execute arbitrary code as users on the local system, by the action of sending a specially-prepared email. While *known* problems have been fixed, alpine code is written in an insecure style and the FreeBSD Security Officer believes there are likely to be other undiscovered vulnerabilities. Users install alpine at their own risk.

The current version of alpine may be installed using the mail/alpine port. Once the port has installed, alpine can be started by issuing the following command:

```
% alpine
```

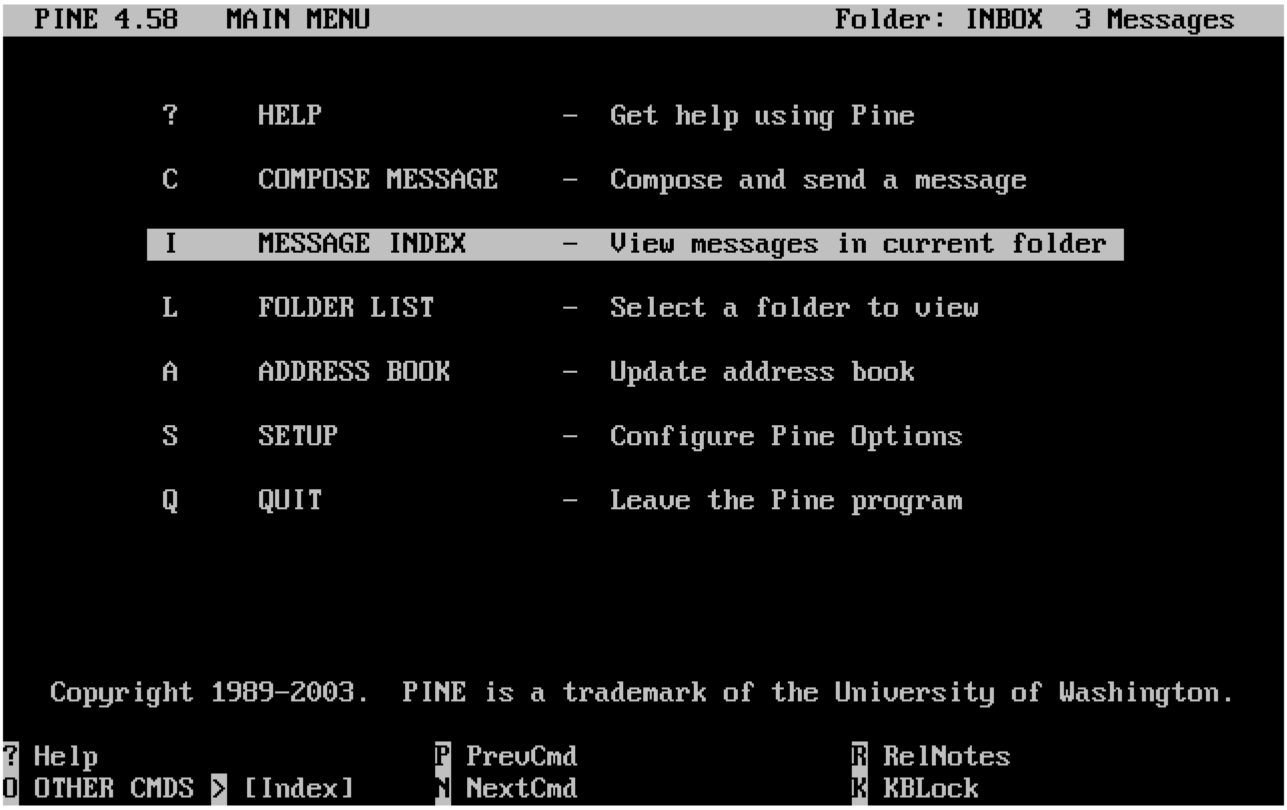
The first time alpine runs, it displays a greeting page with a brief introduction, as well as a request from the alpine development team to send an anonymous email message allowing them to judge how many users are using their client. To send this anonymous message, press `Enter`. Alternatively, press `E` to exit the greeting without sending an anonymous message. An example of the greeting page is shown below:



```
  PINE 4.58   GREETING TEXT                                          No Messages

                    <<<This message will appear only once>>>

           Welcome to Pine ... a Program for Internet News and Email

We hope you will explore Pine's many capabilities. From the Main Menu,
select Setup/Config to see many of the options available to you. Also
note that all screens have context-sensitive help text available.

SPECIAL REQUEST: This software is made available world-wide as a public
service of the University of Washington in Seattle. In order to justify
continuing development, it is helpful to have an idea of how many people
are using Pine. Are you willing to be counted as a Pine user? Pressing
Return will send an anonymous (meaning, your real email address will not
be revealed) message to the Pine development team at the University of
Washington for purposes of tallying.

              Pine is a trademark of the University of Washington.



                             [ALL of greeting text]
? Help      E Exit this greeting                 - PrevPage % Print
          Ret [Be Counted!]                    Spc NextPage
```

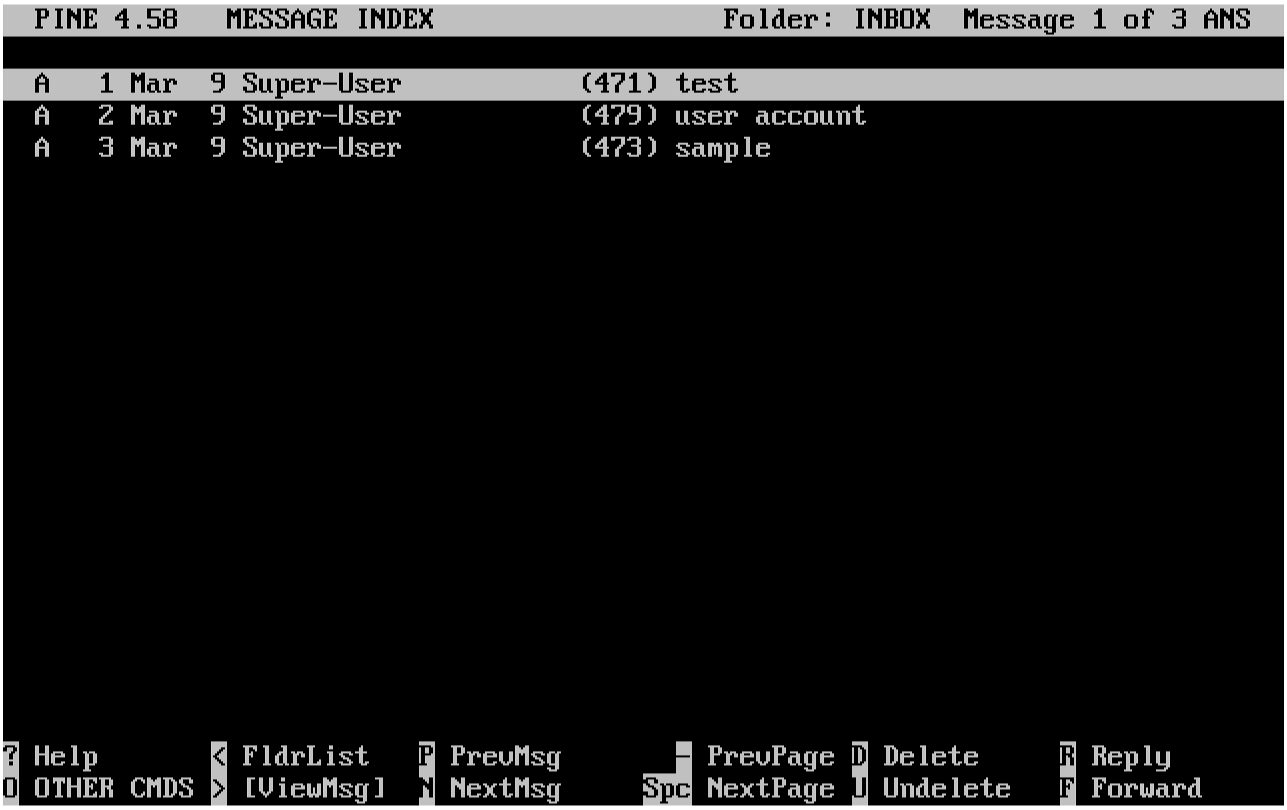
The main menu is then presented, which can be navigated using the cursor keys. This main menu provides shortcuts for the composing new mails, browsing mail directories, and administering address book entries. Below the main menu, relevant keyboard shortcuts to perform functions specific to the task at hand are shown.

The default directory opened by alpine is inbox. To view the message index, press `I`, or select the MESSAGE INDEX option shown below:



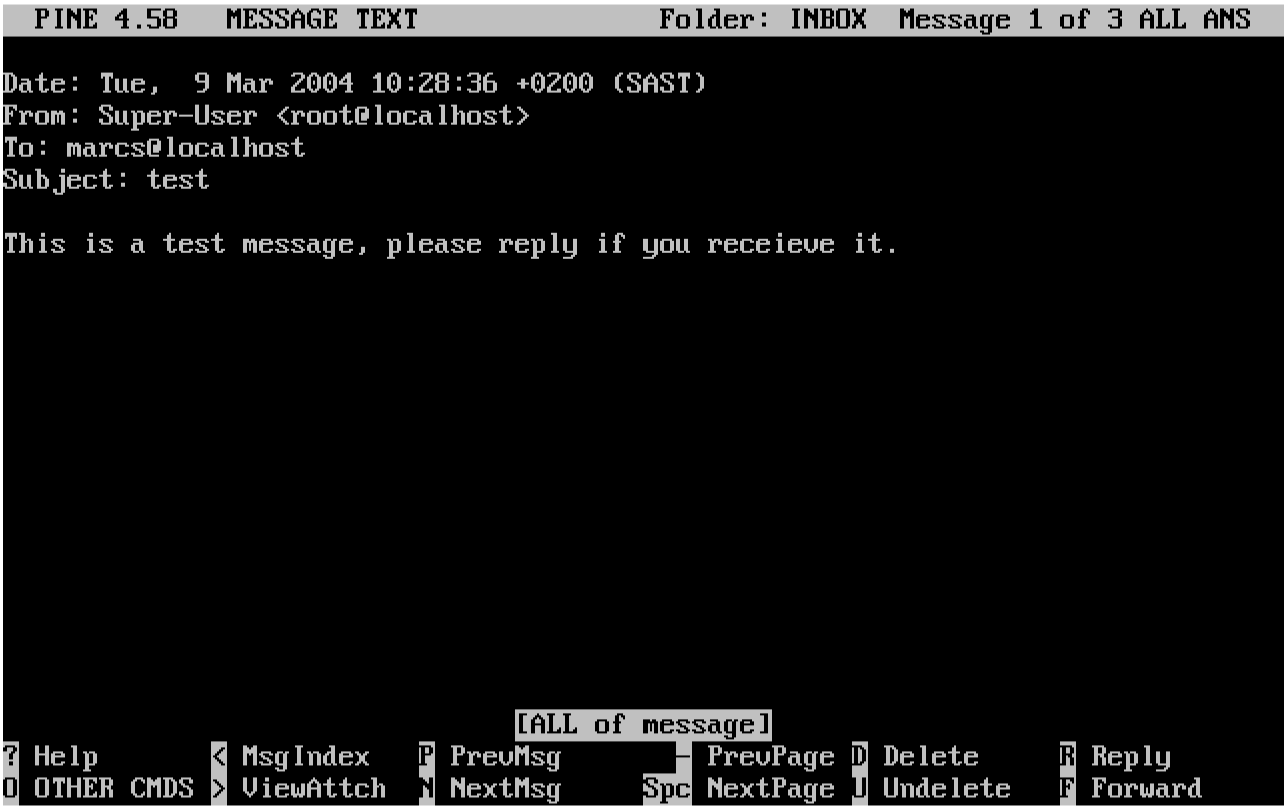
```
  PINE 4.58   MAIN MENU                               Folder: INBOX  3 Messages


          ?     HELP               -  Get help using Pine

          C     COMPOSE MESSAGE    -  Compose and send a message

          I     MESSAGE INDEX      -  View messages in current folder

          L     FOLDER LIST        -  Select a folder to view

          A     ADDRESS BOOK       -  Update address book

          S     SETUP              -  Configure Pine Options

          Q     QUIT               -  Leave the Pine program





   Copyright 1989-2003.  PINE is a trademark of the University of Washington.

? Help                     P PrevCmd                  R RelNotes
O OTHER CMDS > [Index]     N NextCmd                  K KBLock
```

The message index shows messages in the current directory and can be navigated by using the cursor keys. Highlighted messages can be read by pressing `Enter`.



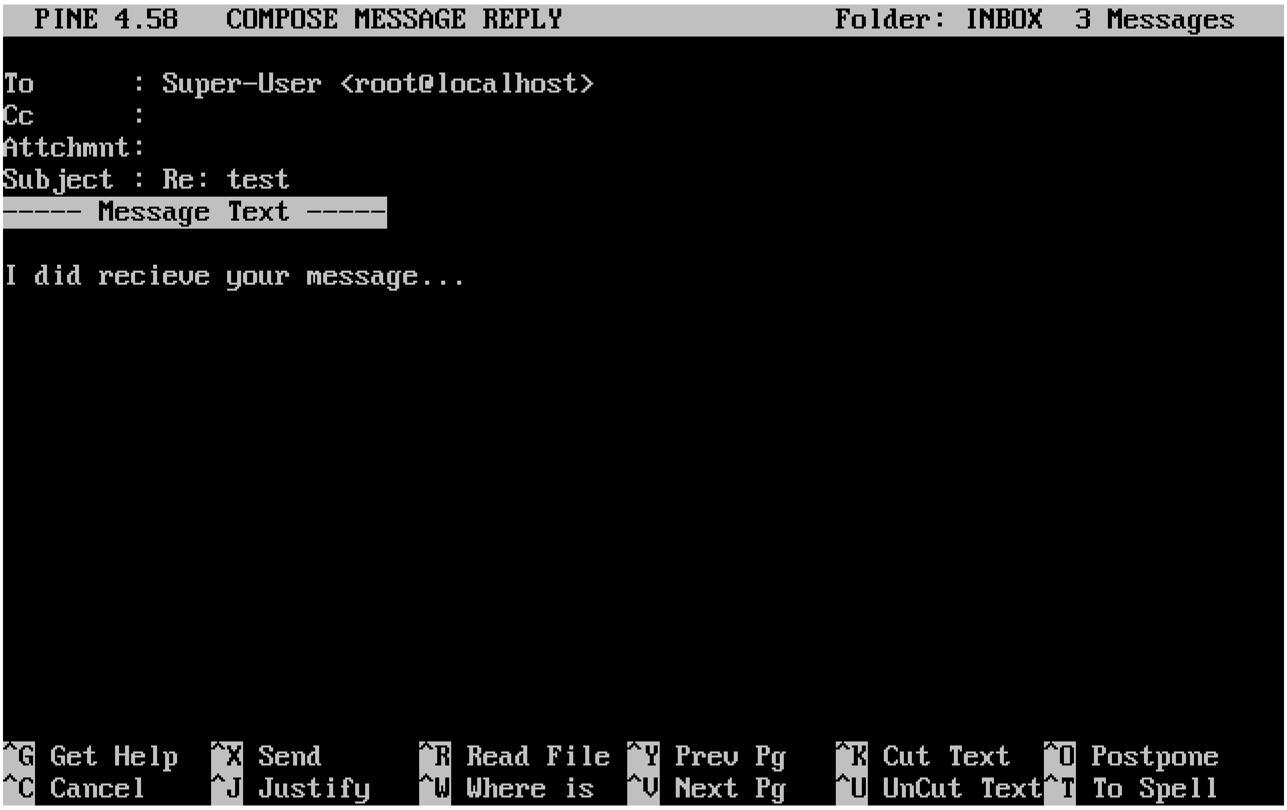
```
  PINE 4.58   MESSAGE INDEX                  Folder: INBOX  Message 1 of 3 ANS

  A   1 Mar  9 Super-User           (471) test
  A   2 Mar  9 Super-User           (479) user account
  A   3 Mar  9 Super-User           (473) sample

? Help         < FldrList    P PrevMsg       - PrevPage D Delete      R Reply
O OTHER CMDS   > [ViewMsg]   N NextMsg     Spc NextPage U Undelete    F Forward
```

In the screenshot below, a sample message is displayed by alpine. Contextual keyboard shortcuts are displayed at the bottom of the screen. An example of one of a shortcut is `r`, which tells the MUA to reply to the current message being displayed.



```
  PINE 4.58   MESSAGE TEXT               Folder: INBOX  Message 1 of 3 ALL ANS

Date: Tue,  9 Mar 2004 10:28:36 +0200 (SAST)
From: Super-User <root@localhost>
To: marcs@localhost
Subject: test

This is a test message, please reply if you receieve it.

                                [ALL of message]
? Help         < MsgIndex    P PrevMsg       - PrevPage D Delete      R Reply
O OTHER CMDS   > ViewAttch   N NextMsg     Spc NextPage U Undelete    F Forward
```

Replying to an email in alpine is done using the pico editor, which is installed by default with alpine. pico makes it easy to navigate the message and is easier for novice users to use than vi(1) or mail(1). Once the reply is complete, the message can be sent by pressing `Ctrl` + `X`. alpine will ask for confirmation before sending the message.



```
  PINE 4.58   COMPOSE MESSAGE REPLY                 Folder: INBOX  3 Messages

To      : Super-User <root@localhost>
Cc      :
Attchmnt:
Subject : Re: test
----- Message Text -----

I did recieve your message...














^G Get Help  ^X Send      ^R Read File ^Y Prev Pg   ^K Cut Text  ^O Postpone
^C Cancel    ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

alpine can be customized using the SETUP option from the main menu.

32.7. Advanced Topics

This section covers more involved topics such as mail configuration and setting up mail for an entire domain.

32.7.1. Basic Configuration

Out of the box, one can send email to external hosts as long as /etc/resolv.conf is configured or the network has access to a configured DNS server. To have email delivered to the MTA on the FreeBSD host, do one of the following:

- Run a DNS server for the domain.
- Get mail delivered directly to the FQDN for the machine.

In order to have mail delivered directly to a host, it must have a permanent static IP address, not a dynamic IP address. If the system is behind a firewall, it must be configured to allow SMTP traffic. To receive mail directly at a host, one of these two must be configured:

- Make sure that the lowest-numbered MX record in DNS points to the host’s static IP address.
- Make sure there is no MX entry in the DNS for the host.

Either of the above will allow mail to be received directly at the host.

Try this:

```
# hostname
```

The output should be similar to the following:

```
example.FreeBSD.org
```

```
# host example.FreeBSD.org
```

The output should be similar to the following:

```
example.FreeBSD.org has address 204.216.27.XX
```

In this example, mail sent directly to yourlogin@example.FreeBSD.org should work without problems, assuming a full-featured MTA is running correctly on `example.FreeBSD.org`. Note that dma(8) does not listen on port 25 for incoming connections and cannot be used in this scenario.

For this example:

```
# host example.FreeBSD.org
```

The output should be similar to the following:

```
example.FreeBSD.org has address 204.216.27.XX
example.FreeBSD.org mail is handled (pri=10) by nevdull.FreeBSD.org
```

All mail sent to `example.FreeBSD.org` will be collected on `nevdull` under the same username instead of being sent directly to the local host.

The above information is handled by the DNS server. The DNS record that carries mail routing information is the mail exchanger record (MX record). If no MX record exists, mail will be delivered directly to the host by way of its IP address.

The MX entry for `freefall.FreeBSD.org` at one time looked like this:

```
freefall        MX  30  mail.crl.net
freefall        MX  40  agora.rdrop.com
freefall        MX  10  freefall.FreeBSD.org
freefall        MX  20  who.cdrom.com
```

`freefall` had many MX entries. The lowest MX number is the host that receives mail directly, if available. If it is not accessible for some reason, the next lower-numbered host will accept messages temporarily, and pass it along when a lower-numbered host becomes available.

Alternate MX sites should have separate Internet connections in order to be most useful. The ISP can provide this service.

32.7.2. Mail for a Domain

When configuring an MTA for a network, any mail sent to hosts in its domain should be diverted to the MTA so that users can receive their mail on the master mail server.

To make life easiest, a user account with the same *username* should exist on both the MTA and the system with the MUA. Use adduser(8) to create the user accounts.

In addition to adding local users to the host, there are alternative methods known as virtual users. Programs like Cyrus and Dovecot can be integrated into MTAs to handle users, mail storage, and also provide access via POP3 and IMAP.

The MTA must be the designated mail exchanger for each workstation on the network. This is done in the DNS configuration with an MX record:

```
example.FreeBSD.org A   204.216.27.XX       ; Workstation
            MX  10 nevdull.FreeBSD.org  ; Mailhost
```

This will redirect mail for the workstation to the MTA no matter where the A record points. The mail is sent to the MX host.

This must be configured on a DNS server. If the network does not run its own DNS server, talk to the ISP or DNS provider.

The following is an example of virtual email hosting.

Consider a customer with the domain `customer1.org`, where all the mail for `customer1.org` should be sent to `mail.myhost.com`.

The DNS entry should look like this:

```
customer1.org       MX  10  mail.myhost.com
```

An `A` record is *not* needed for `customer1.org` in order to only handle email for that domain. However, running `ping` against `customer1.org` will not work unless an `A` record exists for it.

Tell the MTA which domains and/or hostnames it should accept mail for. Either of the following will work for Sendmail:

- Add the hosts to /etc/mail/local-host-names when using the `FEATURE(use_cw_file)`.
- Add a `Cwyour.host.com` line to /etc/sendmail.cf.

32.7.3. Setting Up to Send Only

There are many instances where one may only want to send mail through a relay. Some examples are:

- The computer is a desktop machine that needs to use programs such as mail(1), using the ISP's mail relay.
- The computer is a server that does not handle mail locally, but needs to pass off all mail to a relay for processing.

While any MTA is capable of filling this particular niche, it can be difficult to properly configure a full-featured MTA just to handle offloading mail. Programs such as Sendmail and Postfix are overkill for this use.

Additionally, a typical Internet access service agreement may forbid one from running a "mail server".

The easiest way to fulfill those needs is to use the dma(8) MTA included in the base system. For systems up to 13.2, need be to installed from ports.

In addition to dma(8), third-party software can be used to achieve the same, like mail/ssmtp.

```
# cd /usr/ports/mail/ssmtp
# make install replace clean
```

Once installed, mail/ssmtp can be configured with /usr/local/etc/ssmtp/ssmtp.conf:

```
root=yourrealemail@example.com
mailhub=mail.example.com
rewriteDomain=example.com
hostname=_HOSTNAME_
```

Use the real email address for `root`. Enter the ISP's outgoing mail relay in place of `mail.example.com`. Some ISPs call this the "outgoing mail server" or "SMTP server".

Make sure to disable Sendmail, including the outgoing mail service.

mail/ssmtp has some other options available. Refer to the examples in /usr/local/etc/ssmtp or the manual page of ssmtp for more information.

Setting up ssmtp in this manner allows any software on the computer that needs to send mail to function properly, while not violating the ISP's usage policy or allowing the computer to be hijacked for spamming.

32.7.4. SMTP Authentication in Sendmail

Configuring SMTP authentication on the MTA provides a number of benefits. SMTP authentication adds a layer of security to Sendmail, and provides mobile users who switch hosts the ability to use the same MTA without the need to reconfigure their mail client's settings each time.

Install security/cyrus-sasl2 from the Ports Collection. This port supports a number of compile-time options. For the SMTP authentication method demonstrated in this example, make sure that `LOGIN` is not disabled.

After installing security/cyrus-sasl2, edit /usr/local/lib/sasl2/Sendmail.conf, or create it if it does not exist, and add the following line:

```
pwcheck_method: saslauthd
```

Next, install security/cyrus-sasl2-saslauthd and add execute the following command:

```
# sysrc saslauthd_enable="YES"
```

Finally, start the saslauthd daemon:

```
# service saslauthd start
```

This daemon serves as a broker for Sendmail to authenticate against the FreeBSD passwd(5) database. This saves the trouble of creating a new set of usernames and passwords for each user that needs to use SMTP authentication, and keeps the login and mail password the same.

Next, edit /etc/make.conf and add the following lines:

```
SENDMAIL_CFLAGS=-I/usr/local/include/sasl -DSASL
SENDMAIL_LDADD=/usr/local/lib/libsasl2.so
```

These lines provide Sendmail the proper configuration options for linking to cyrus-sasl2 at compile time. Make sure that cyrus-sasl2 has been installed before recompiling Sendmail.

Recompile Sendmail by executing the following commands:

```
# cd /usr/src/lib/libsmutil
# make cleandir && make obj && make
# cd /usr/src/lib/libsm
# make cleandir && make obj && make
# cd /usr/src/usr.sbin/sendmail
# make cleandir && make obj && make && make install
```

This compile should not have any problems if /usr/src has not changed extensively and the shared libraries it needs are available.

After Sendmail has been compiled and reinstalled, edit /etc/mail/freebsd.mc or the local .mc. Many administrators choose to use the output from hostname(1) as the name of .mc for uniqueness.

Add these lines:

```
dnl set SASL options
TRUST_AUTH_MECH(`GSSAPI DIGEST-MD5 CRAM-MD5 LOGIN')dnl
define(`confAUTH_MECHANISMS', `GSSAPI DIGEST-MD5 CRAM-MD5 LOGIN')dnl
```

These options configure the different methods available to Sendmail for authenticating users. To use a method other than pwcheck, refer to the Sendmail documentation.

Finally, run make(1) while in /etc/mail. That will run the new .mc and create a .cf named either freebsd.cf or the name used for the local .mc.

Then, run `make install restart`, which will copy the file to sendmail.cf, and properly restart Sendmail.

For more information about this process, refer to /etc/mail/Makefile.

To test the configuration, use a MUA to send a test message. For further investigation, set the `LogLevel` of Sendmail to `13` and watch /var/log/maillog for any errors.

For more information, refer to SMTP authentication.

Chapter 33. Network Servers

33.1. Synopsis

This chapter covers some of the more frequently used network services on UNIX® systems. This includes installing, configuring, testing, and maintaining many different types of network services. Example configuration files are included throughout this chapter for reference.

By the end of this chapter, readers will know:

- How to manage the inetd daemon.
- How to set up the Network File System (NFS).
- How to set up the Network Information Server (NIS) for centralizing and sharing user accounts.
- How to set FreeBSD up to act as an LDAP server or client
- How to set up automatic network settings using DHCP.
- How to set up a Domain Name Server (DNS).
- How to set up the Apache HTTP Server.
- How to set up a File Transfer Protocol (FTP) server.
- How to set up a file and print server for Windows® clients using Samba.
- How to synchronize the time and date, and set up a time server using the Network Time Protocol (NTP).
- How to set up iSCSI.

This chapter assumes a basic knowledge of:

- /etc/rc scripts.
- Network terminology.
- Installation of additional third-party software (Installing Applications: Packages and Ports).

33.2. The inetd Super-Server

The inetd(8) daemon is sometimes referred to as a Super-Server because it manages connections for many services. Instead of starting multiple applications, only the inetd service needs to be started. When a connection is received for a service that is managed by inetd, it determines which program the connection is destined for, spawns a process for that program, and delegates the program a socket. Using inetd for services that are not heavily used can reduce system load, when compared to running each daemon individually in stand-alone mode.

Primarily, inetd is used to spawn other daemons, but several trivial protocols are handled internally, such as chargen, auth, time, echo, discard, and daytime.

This section covers the basics of configuring inetd.

33.2.1. Configuration File

Configuration of inetd is done by editing /etc/inetd.conf. Each line of this configuration file represents an application which can be started by inetd. By default, every line starts with a comment (`#`), meaning that inetd is not listening for any applications. To configure inetd to listen for an application's connections, remove the `#` at the beginning of the line for that application.

After saving the edits, configure inetd to start at system boot by editing /etc/rc.conf:

```
inetd_enable="YES"
```

To start inetd now, so that it listens for the configured service, type:

```
# service inetd start
```

Once inetd is started, it needs to be notified whenever a modification is made to /etc/inetd.conf:

Example 41. Reloading the inetd Configuration File

```
# service inetd reload
```

Typically, the default entry for an application does not need to be edited beyond removing the `#`. In some situations, it may be appropriate to edit the default entry.

As an example, this is the default entry for ftpd(8) over IPv4:

```
ftp     stream  tcp     nowait  root    /usr/libexec/ftpd       ftpd -l
```

The seven columns in an entry are as follows:

```
service-name
socket-type
protocol
{wait|nowait}[/max-child[/max-connections-per-ip-per-minute[/max-child-per-ip]]]
user[:group][/login-class]
server-program
server-program-arguments
```

where:

service-name

The service name of the daemon to start. It must correspond to a service listed in /etc/services. This determines which port inetd listens on for incoming connections to that service. When using a custom service, it must first be added to /etc/services.

socket-type

Either `stream`, `dgram`, `raw`, or `seqpacket`. Use `stream` for TCP connections and `dgram` for UDP services.

protocol

Use one of the following protocol names:

Protocol Name	Explanation
tcp or tcp4	TCP IPv4
udp or udp4	UDP IPv4
tcp6	TCP IPv6
udp6	UDP IPv6
tcp46	Both TCP IPv4 and IPv6
udp46	Both UDP IPv4 and IPv6

{wait|nowait}[/max-child[/max-connections-per-ip-per-minute[/max-child-per-ip]]]

In this field, `wait` or `nowait` must be specified. `max-child`, `max-connections-per-ip-per-minute` and `max-child-per-ip` are optional.

`wait|nowait` indicates whether or not the service is able to handle its own socket. `dgram` socket types must use `wait` while `stream` daemons, which are usually multi-threaded, should use `nowait`. `wait` usually hands off multiple sockets to a single daemon, while `nowait` spawns a child daemon for each new socket.

The maximum number of child daemons inetd may spawn is set by `max-child`. For example, to limit ten instances of the daemon, place a `/10` after `nowait`. Specifying `/0` allows an unlimited number of children.

`max-connections-per-ip-per-minute` limits the number of connections from any particular IP address per minute. Once the limit is reached, further connections from this IP address will be dropped until the end of the minute. For example, a value of `/10` would limit any particular IP address to ten connection attempts per minute. `max-child-per-ip` limits the number of child processes that can be started on behalf on any single IP address at any moment. These options can limit excessive resource consumption and help to prevent Denial of Service attacks.

An example can be seen in the default settings for fingerd(8):

```
finger stream  tcp    nowait/3/10 nobody /usr/libexec/fingerd fingerd -k -s
```

user

The username the daemon will run as. Daemons typically run as `root`, `daemon`, or `nobody`.

server-program

The full path to the daemon. If the daemon is a service provided by inetd internally, use `internal`.

server-program-arguments

Used to specify any command arguments to be passed to the daemon on invocation. If the daemon is an internal service, use `internal`.

33.2.2. Command-Line Options

Like most server daemons, inetd has a number of options that can be used to modify its behavior. By default, inetd is started with `-wW -C 60`. These options enable TCP wrappers for all services, including internal services, and prevent any IP address from requesting any service more than 60 times per minute.

To change the default options which are passed to inetd, add an entry for `inetd_flags` in /etc/rc.conf. If inetd is already running, restart it with `service inetd restart`.

The available rate limiting options are:

-c maximum

Specify the default maximum number of simultaneous invocations of each service, where the default is unlimited. May be overridden on a per-service basis by using `max-child` in /etc/inetd.conf.

-C rate

Specify the default maximum number of times a service can be invoked from a single IP address per minute. May be overridden on a per-service basis by using `max-connections-per-ip-per-minute` in /etc/inetd.conf.

-R rate

Specify the maximum number of times a service can be invoked in one minute, where the default is `256`. A rate of `0` allows an unlimited number.

-s maximum

Specify the maximum number of times a service can be invoked from a single IP address at any one time, where the default is unlimited. May be overridden on a per-service basis by using `max-child-per-ip` in /etc/inetd.conf.

Additional options are available. Refer to inetd(8) for the full list of options.

33.2.3. Security Considerations

Many of the daemons which can be managed by inetd are not security-conscious. Some daemons, such as fingerd, can provide information that may be useful to an attacker. Only enable the services which are needed and monitor the system for excessive connection attempts. `max-connections-per-ip-per-minute`, `max-child` and `max-child-per-ip` can be used to limit such attacks.

By default, TCP wrappers are enabled. Consult hosts_access(5) for more information on placing TCP restrictions on various inetd invoked daemons.

33.3. Network File System (NFS)

FreeBSD supports the Network File System (NFS), which allows a server to share directories and files with clients over a network. With NFS, users and programs can access files on remote systems as if they were stored locally.

NFS has many practical uses. Some of the more common uses include:

- Data that would otherwise be duplicated on each client can be kept in a single location and accessed by clients on the network.
- Several clients may need access to the /usr/ports/distfiles directory. Sharing that directory allows for quick access to the source files without having to download them to each client.
- On large networks, it is often more convenient to configure a central NFS server on which all user home directories are stored. Users can log into a client anywhere on the network and have access to their home directories.
- Administration of NFS exports is simplified. For example, there is only one file system where security or backup policies must be set.
- Removable media storage devices can be used by other machines on the network. This reduces the number of devices throughout the network and provides a centralized location to manage their security. It is often more convenient to install software on multiple machines from a centralized installation media.

NFS consists of a server and one or more clients. The client remotely accesses the data that is stored on the server machine. In order for this to function properly, a few processes have to be configured and running.

These daemons must be running on the server:

Daemon	**Description**
nfsd	The NFS daemon which services requests from NFS clients.
mountd	The NFS mount daemon which carries out requests received from nfsd.
rpcbind	This daemon allows NFS clients to discover which port the NFS server is using.

Running nfsiod(8) on the client can improve performance, but is not required.

33.3.1. Configuring the Server

The file systems which the NFS server will share are specified in /etc/exports. Each line in this file specifies a file system to be exported, which clients have access to that file system, and any access options. When adding entries to this file, each exported file system, its properties, and allowed hosts must occur on a single line. If no clients are listed in the entry, then any client on the network can mount that file system.

The following /etc/exports entries demonstrate how to export file systems. The examples can be modified to match the file systems and client names on the reader's network. There are many options that can be used in this file, but only a few will be mentioned here. See exports(5) for the full list of options.

This example shows how to export /media to three hosts named *alpha*, *bravo*, and *charlie*:

```
/media -ro alpha bravo charlie
```

The `-ro` flag makes the file system read-only, preventing clients from making any changes to the exported file system. This example assumes that the host names are either in DNS or in /etc/hosts. Refer to hosts(5) if the network does not have a DNS server.

The next example exports /home to three clients by IP address. This can be useful for networks without DNS or /etc/hosts entries. The `-alldirs` flag allows subdirectories to be mount points. In other words, it will not automatically mount the subdirectories, but will permit the client to mount the directories that are required as needed.

```
/usr/home  -alldirs  10.0.0.2 10.0.0.3 10.0.0.4
```

This next example exports /a so that two clients from different domains may access that file system. The `-maproot=root` allows `root` on the remote system to write data on the exported file system as `root`. If `-maproot=root` is not specified, the client's `root` user will be mapped to the server's `nobody` account and will be subject to the access limitations defined for `nobody`.

```
/a  -maproot=root  host.example.com box.example.org
```

A client can only be specified once per file system. For example, if /usr is a single file system, these entries would be invalid as both entries specify the same host:

```
# Invalid when /usr is one file system
/usr/src   client
/usr/ports client
```

The correct format for this situation is to use one entry:

```
/usr/src /usr/ports  client
```

The following is an example of a valid export list, where /usr and /exports are local file systems:

```
# Export src and ports to client01 and client02, but only
# client01 has root privileges on it
/usr/src /usr/ports -maproot=root    client01
/usr/src /usr/ports               client02
```

```
# The client machines have root and can mount anywhere
# on /exports. Anyone in the world can mount /exports/obj read-only
/exports -alldirs -maproot=root      client01 client02
/exports/obj -ro
```

To enable the processes required by the NFS server at boot time, add these options to /etc/rc.conf:

```
rpcbind_enable="YES"
nfs_server_enable="YES"
mountd_enable="YES"
```

The server can be started now by running this command:

```
# service nfsd start
```

Whenever the NFS server is started, mountd also starts automatically. However, mountd only reads /etc/exports when it is started. To make subsequent /etc/exports edits take effect immediately, force mountd to reread it:

```
# service mountd reload
```

Refer to zfs-share(8) for a description of exporting ZFS datasets via NFS using the `sharenfs` ZFS property instead of the exports(5) file.

Refer to nfsv4(4) for a description of an NFS Version 4 setup.

33.3.2. Configuring the Client

To enable NFS clients, set this option in each client's /etc/rc.conf:

```
nfs_client_enable="YES"
```

Then, run this command on each NFS client:

```
# service nfsclient start
```

The client now has everything it needs to mount a remote file system. In these examples, the server's name is `server` and the client's name is `client`. To mount /home on `server` to the /mnt mount point on `client`:

```
# mount server:/home /mnt
```

The files and directories in /home will now be available on `client`, in the /mnt directory.

To mount a remote file system each time the client boots, add it to /etc/fstab:

```
server:/home    /mnt    nfs rw  0   0
```

Refer to fstab(5) for a description of all available options.

33.3.3. Locking

Some applications require file locking to operate correctly. To enable locking, execute the following command on both the client and server:

```
# sysrc rpc_lockd_enable="YES"
```

Then start the rpc.lockd(8) service:

```
# service lockd start
```

If locking is not required on the server, the NFS client can be configured to lock locally by including `-L` when running mount. Refer to mount_nfs(8) for further details.

33.3.4. Automating Mounts with autofs(5)

The autofs(5) automount facility is supported starting with FreeBSD 10.1-RELEASE. To use the automounter functionality in older versions of FreeBSD, use amd(8) instead. This chapter only describes the autofs(5) automounter.

The autofs(5) facility is a common name for several components that, together, allow for automatic mounting of remote and local filesystems whenever a file or directory within that file system is accessed. It consists of the kernel component, autofs(5), and several userspace applications: automount(8), automountd(8) and autounmountd(8). It serves as an alternative for amd(8) from previous FreeBSD releases. amd is still provided for backward compatibility purposes, as the two use different map formats; the one used by autofs is the same as with other SVR4 automounters, such as the ones in Solaris, MacOS X, and Linux.

The autofs(5) virtual filesystem is mounted on specified mountpoints by automount(8), usually invoked during boot.

Whenever a process attempts to access a file within the autofs(5) mountpoint, the kernel will notify automountd(8) daemon and pause the triggering process. The automountd(8) daemon will handle kernel requests by finding the proper map and mounting the filesystem according to it, then signal the kernel to release blocked process. The autounmountd(8) daemon automatically unmounts automounted filesystems after some time, unless they are still being used.

The primary autofs configuration file is /etc/auto_master. It assigns individual maps to top-level mounts. For an explanation of auto_master and the map syntax, refer to auto_master(5).

There is a special automounter map mounted on /net. When a file is accessed within this directory, autofs(5) looks up the corresponding remote mount and automatically mounts it. For instance, an attempt to access a file within /net/foobar/usr would tell automountd(8) to mount the /usr export from the host `foobar`.

Example 42. Mounting an Export with autofs(5)

In this example, `showmount -e` shows the exported file systems that can be mounted from the NFS server, `foobar`:

```
% showmount -e foobar
Exports list on foobar:
/usr                               10.10.10.0
/a                                 10.10.10.0
% cd /net/foobar/usr
```

The output from `showmount` shows /usr as an export. When changing directories to /host/foobar/usr, automountd(8) intercepts the request and attempts to resolve the hostname `foobar`. If successful, automountd(8) automatically mounts the source export.

To enable autofs(5) at boot time, add this line to /etc/rc.conf:

```
autofs_enable="YES"
```

Then autofs(5) can be started by running:

```
# service automount start
# service automountd start
# service autounmountd start
```

The autofs(5) map format is the same as in other operating systems. Information about this format from other sources can be useful, like the Mac OS X document.

Consult the automount(8), automountd(8), autounmountd(8), and auto_master(5) manual pages for more information.

33.4. Network Information System (NIS)

Network Information System (NIS) is designed to centralize administration of UNIX®-like systems such as Solaris™, HP-UX, AIX®, Linux, NetBSD, OpenBSD, and FreeBSD. NIS was originally known as Yellow Pages but the name was changed due to trademark issues. This is the reason why NIS commands begin with `yp`.

NIS is a Remote Procedure Call (RPC)-based client/server system that allows a group of machines within an NIS domain to share a common set of configuration files. This permits a system administrator to set up NIS client systems with only minimal configuration data and to add, remove, or modify configuration data from a single location.

FreeBSD uses version 2 of the NIS protocol.

33.4.1. NIS Terms and Processes

Table 28.1 summarizes the terms and important processes used by NIS:

Table 45. NIS Terminology

Term	Description
NIS domain name	NIS servers and clients share an NIS domain name. Typically, this name does not have anything to do with DNS.
rpcbind(8)	This service enables RPC and must be running in order to run an NIS server or act as an NIS client.
ypbind(8)	This service binds an NIS client to its NIS server. It will take the NIS domain name and use RPC to connect to the server. It is the core of client/server communication in an NIS environment. If this service is not running on a client machine, it will not be able to access the NIS server.
ypserv(8)	This is the process for the NIS server. If this service stops running, the server will no longer be able to respond to NIS requests so hopefully, there is a slave server to take over. Some non-FreeBSD clients will not try to reconnect using a slave server and the ypbind process may need to be restarted on these clients.
rpc.yppasswdd(8)	This process only runs on NIS master servers. This daemon allows NIS clients to change their NIS passwords. If this daemon is not running, users will have to login to the NIS master server and change their passwords there.

33.4.2. Machine Types

There are three types of hosts in an NIS environment:

- NIS master server

  This server acts as a central repository for host configuration information and maintains the authoritative copy of the files used by all of the NIS clients. The passwd, group, and other various files used by NIS clients are stored on the master server. While it is possible for one machine to be an NIS master server for more than one NIS domain, this type of configuration will not be covered in this chapter as it assumes a relatively small-scale NIS environment.

- NIS slave servers

  NIS slave servers maintain copies of the NIS master's data files in order to provide redundancy. Slave servers also help to balance the load of the master server as NIS clients always attach to the NIS server which responds first.

- NIS clients

  NIS clients authenticate against the NIS server during log on.

Information in many files can be shared using NIS. The master.passwd, group, and hosts files are commonly shared via NIS. Whenever a process on a client needs information that would normally be found in these files locally, it makes a query to the NIS server that it is bound to instead.

33.4.3. Planning Considerations

This section describes a sample NIS environment which consists of 15 FreeBSD machines with no centralized point of administration. Each machine has its own /etc/passwd and /etc/master.passwd. These files are kept in sync with each other only through manual intervention. Currently, when a user is added to the lab, the process must be repeated on all 15 machines.

The configuration of the lab will be as follows:

Machine name	IP address	Machine role
`ellington`	`10.0.0.2`	NIS master
`coltrane`	`10.0.0.3`	NIS slave
`basie`	`10.0.0.4`	Faculty workstation
`bird`	`10.0.0.5`	Client machine
`cli[1-11]`	`10.0.0.[6-17]`	Other client machines

If this is the first time an NIS scheme is being developed, it should be thoroughly planned ahead of time. Regardless of network size, several decisions need to be made as part of the planning process.

Choosing a NIS Domain Name

When a client broadcasts its requests for info, it includes the name of the NIS domain that it is part of. This is how multiple servers on one network can tell which server should answer which request. Think of the NIS domain name as the name for a group of hosts.

Some organizations choose to use their Internet domain name for their NIS domain name. This is not recommended as it can cause confusion when trying to debug network problems. The NIS domain name should be unique within the network and it is helpful if it describes the group of machines it represents. For example, the Art department at Acme Inc. might be in the "acme-art" NIS domain. This example will use the domain name `test-domain`.

However, some non-FreeBSD operating systems require the NIS domain name to be the same as the Internet domain name. If one or more machines on the network have this restriction, the Internet domain name *must* be used as the NIS domain name.

Physical Server Requirements

There are several things to keep in mind when choosing a machine to use as a NIS server. Since NIS clients depend upon the availability of the server, choose a machine that is not rebooted frequently. The NIS server should ideally be a stand alone machine whose sole purpose is to be an NIS server. If the network is not heavily used, it is acceptable to put the NIS server on a machine running other services. However, if the NIS server becomes unavailable, it will adversely affect all NIS clients.

33.4.4. Configuring the NIS Master Server

The canonical copies of all NIS files are stored on the master server. The databases used to store the information are called NIS maps. In FreeBSD, these maps are stored in /var/yp/[domainname] where [domainname] is the name of the NIS domain. Since multiple domains are supported, it is possible to have several directories, one for each domain. Each domain will have its own independent set of maps.

NIS master and slave servers handle all NIS requests through ypserv(8). This daemon is responsible for receiving incoming requests from NIS clients, translating the requested domain and map name to a path to the corresponding database file, and transmitting data from the database back to the client.

Setting up a master NIS server can be relatively straight forward, depending on environmental needs. Since FreeBSD provides built-in NIS support, it only needs to be enabled by adding the following lines to /etc/rc.conf:

```
nisdomainname="test-domain" ①
nis_server_enable="YES"     ②
nis_yppasswdd_enable="YES"  ③
```

① This line sets the NIS domain name to `test-domain`.

② This automates the start up of the NIS server processes when the system boots.

③ This enables the rpc.yppasswdd(8) daemon so that users can change their NIS password from a client machine.

Care must be taken in a multi-server domain where the server machines are also NIS clients. It is generally a good idea to force the servers to bind to themselves rather than allowing them to broadcast bind requests and possibly become bound to each other. Strange failure modes can result if one server goes down and others are dependent upon it. Eventually, all the clients will time out and attempt to bind to other servers, but the delay involved can be considerable and the failure mode is still present since the servers might bind to each other all over again.

A server that is also a client can be forced to bind to a particular server by adding these additional lines to /etc/rc.conf:

```
nis_client_enable="YES"          ①
nis_client_flags="-S test-domain,server"    ②
```

① This enables running client stuff as well.

② This line sets the NIS domain name to `test-domain` and bind to itself.

After saving the edits, type `/etc/netstart` to restart the network and apply the values defined in /etc/rc.conf. Before initializing the NIS maps, start ypserv(8):

```
# service ypserv start
```

Initializing the NIS Maps

NIS maps are generated from the configuration files in /etc on the NIS master, with one exception: /etc/master.passwd. This is to prevent the propagation of passwords to all the servers in the NIS domain. Therefore, before the NIS maps are initialized, configure the primary password files:

```
# cp /etc/master.passwd /var/yp/master.passwd
# cd /var/yp
# vi master.passwd
```

It is advisable to remove all entries for system accounts as well as any user accounts that do not need to be propagated to the NIS clients, such as the `root` and any other administrative accounts.

Ensure that the /var/yp/master.passwd is neither group or world readable by setting its permissions to `600`.

After completing this task, initialize the NIS maps. FreeBSD includes the ypinit(8) script to do this. When generating maps for the master server, include `-m` and specify the NIS domain name:

```
ellington# ypinit -m test-domain
Server Type: MASTER Domain: test-domain
Creating an YP server will require that you answer a few questions.
Questions will all be asked at the beginning of the procedure.
Do you want this procedure to quit on non-fatal errors? [y/n: n] n
Ok, please remember to go back and redo manually whatever fails.
If not, something might not work.
At this point, we have to construct a list of this domains YP servers.
rod.darktech.org is already known as master server.
Please continue to add any slave servers, one per line. When you are
done with the list, type a <control D>.
master server   :  ellington
next host to add:  coltrane
next host to add:  ^D
The current list of NIS servers looks like this:
ellington
```

```
coltrane
Is this correct?  [y/n: y] y

[..output from map generation..]

NIS Map update completed.
ellington has been setup as an YP master server without any errors.
```

This will create /var/yp/Makefile from /var/yp/Makefile.dist. By default, this file assumes that the environment has a single NIS server with only FreeBSD clients. Since `test-domain` has a slave server, edit this line in /var/yp/Makefile so that it begins with a comment (`#`):

```
NOPUSH = "True"
```

Adding New Users

Every time a new user is created, the user account must be added to the master NIS server and the NIS maps rebuilt. Until this occurs, the new user will not be able to login anywhere except on the NIS master. For example, to add the new user `jsmith` to the `test-domain` domain, run these commands on the master server:

```
# pw useradd jsmith
# cd /var/yp
# make test-domain
```

The user could also be added using `adduser jsmith` instead of `pw useradd smith`.

33.4.5. Setting up a NIS Slave Server

To set up an NIS slave server, log on to the slave server and edit /etc/rc.conf as for the master server. Do not generate any NIS maps, as these already exist on the master server. When running `ypinit` on the slave server, use `-s` (for slave) instead of `-m` (for master). This option requires the name of the NIS master in addition to the domain name, as seen in this example:

```
coltrane# ypinit -s ellington test-domain

Server Type: SLAVE Domain: test-domain Master: ellington

Creating an YP server will require that you answer a few questions.
Questions will all be asked at the beginning of the procedure.

Do you want this procedure to quit on non-fatal errors? [y/n: n]  n

Ok, please remember to go back and redo manually whatever fails.
If not, something might not work.
There will be no further questions. The remainder of the procedure
should take a few minutes, to copy the databases from ellington.
```

```
Transferring netgroup...
ypxfr: Exiting: Map successfully transferred
Transferring netgroup.byuser...
ypxfr: Exiting: Map successfully transferred
Transferring netgroup.byhost...
ypxfr: Exiting: Map successfully transferred
Transferring master.passwd.byuid...
ypxfr: Exiting: Map successfully transferred
Transferring passwd.byuid...
ypxfr: Exiting: Map successfully transferred
Transferring passwd.byname...
ypxfr: Exiting: Map successfully transferred
Transferring group.bygid...
ypxfr: Exiting: Map successfully transferred
Transferring group.byname...
ypxfr: Exiting: Map successfully transferred
Transferring services.byname...
ypxfr: Exiting: Map successfully transferred
Transferring rpc.bynumber...
ypxfr: Exiting: Map successfully transferred
Transferring rpc.byname...
ypxfr: Exiting: Map successfully transferred
Transferring protocols.byname...
ypxfr: Exiting: Map successfully transferred
Transferring master.passwd.byname...
ypxfr: Exiting: Map successfully transferred
Transferring networks.byname...
ypxfr: Exiting: Map successfully transferred
Transferring networks.byaddr...
ypxfr: Exiting: Map successfully transferred
Transferring netid.byname...
ypxfr: Exiting: Map successfully transferred
Transferring hosts.byaddr...
ypxfr: Exiting: Map successfully transferred
Transferring protocols.bynumber...
ypxfr: Exiting: Map successfully transferred
Transferring ypservers...
ypxfr: Exiting: Map successfully transferred
Transferring hosts.byname...
ypxfr: Exiting: Map successfully transferred

coltrane has been setup as an YP slave server without any errors.
Remember to update map ypservers on ellington.
```

This will generate a directory on the slave server called /var/yp/test-domain which contains copies of the NIS master server's maps. Adding these /etc/crontab entries on each slave server will force the slaves to sync their maps with the maps on the master server:

```
20      *       *       *       *       root   /usr/libexec/ypxfr passwd.byname
```

```
21      *       *       *       *       root   /usr/libexec/ypxfr passwd.byuid
```

These entries are not mandatory because the master server automatically attempts to push any map changes to its slaves. However, since clients may depend upon the slave server to provide correct password information, it is recommended to force frequent password map updates. This is especially important on busy networks where map updates might not always complete.

To finish the configuration, run `/etc/netstart` on the slave server in order to start the NIS services.

33.4.6. Setting Up an NIS Client

An NIS client binds to an NIS server using ypbind(8). This daemon broadcasts RPC requests on the local network. These requests specify the domain name configured on the client. If an NIS server in the same domain receives one of the broadcasts, it will respond to ypbind, which will record the server's address. If there are several servers available, the client will use the address of the first server to respond and will direct all of its NIS requests to that server. The client will automatically ping the server on a regular basis to make sure it is still available. If it fails to receive a reply within a reasonable amount of time, ypbind will mark the domain as unbound and begin broadcasting again in the hopes of locating another server.

To configure a FreeBSD machine to be an NIS client:

1. Edit /etc/rc.conf and add the following lines in order to set the NIS domain name and start ypbind(8) during network startup:

   ```
   nisdomainname="test-domain"
   nis_client_enable="YES"
   ```

2. To import all possible password entries from the NIS server, use `vipw` to remove all user accounts except one from /etc/master.passwd. When removing the accounts, keep in mind that at least one local account should remain and this account should be a member of `wheel`. If there is a problem with NIS, this local account can be used to log in remotely, become the superuser, and fix the problem. Before saving the edits, add the following line to the end of the file:

   ```
   +:::::::::
   ```

   This line configures the client to provide anyone with a valid account in the NIS server's password maps an account on the client. There are many ways to configure the NIS client by modifying this line. One method is described in Using Netgroups. For more detailed reading, refer to the book `Managing NFS and NIS`, published by O'Reilly Media.

3. To import all possible group entries from the NIS server, add this line to /etc/group:

   ```
   +:*::
   ```

To start the NIS client immediately, execute the following commands as the superuser:

```
# /etc/netstart
# service ypbind start
```

After completing these steps, running `ypcat passwd` on the client should show the server's passwd map.

33.4.7. NIS Security

Since RPC is a broadcast-based service, any system running ypbind within the same domain can retrieve the contents of the NIS maps. To prevent unauthorized transactions, ypserv(8) supports a feature called "securenets" which can be used to restrict access to a given set of hosts. By default, this information is stored in /var/yp/securenets, unless ypserv(8) is started with `-p` and an alternate path. This file contains entries that consist of a network specification and a network mask separated by white space. Lines starting with `"#"` are considered to be comments. A sample securenets might look like this:

```
# allow connections from local host -- mandatory
127.0.0.1     255.255.255.255
# allow connections from any host
# on the 192.168.128.0 network
192.168.128.0 255.255.255.0
# allow connections from any host
# between 10.0.0.0 to 10.0.15.255
# this includes the machines in the testlab
10.0.0.0      255.255.240.0
```

If ypserv(8) receives a request from an address that matches one of these rules, it will process the request normally. If the address fails to match a rule, the request will be ignored and a warning message will be logged. If the securenets does not exist, `ypserv` will allow connections from any host.

TCP Wrapper is an alternate mechanism for providing access control instead of securenets. While either access control mechanism adds some security, they are both vulnerable to "IP spoofing" attacks. All NIS-related traffic should be blocked at the firewall.

Servers using securenets may fail to serve legitimate NIS clients with archaic TCP/IP implementations. Some of these implementations set all host bits to zero when doing broadcasts or fail to observe the subnet mask when calculating the broadcast address. While some of these problems can be fixed by changing the client configuration, other problems may force the retirement of these client systems or the abandonment of securenets.

The use of TCP Wrapper increases the latency of the NIS server. The additional delay may be long enough to cause timeouts in client programs, especially in busy networks with slow NIS servers. If one or more clients suffer from latency, convert those clients into NIS slave servers and force them to bind to themselves.

Barring Some Users

In this example, the `basie` system is a faculty workstation within the NIS domain. The passwd map on the master NIS server contains accounts for both faculty and students. This section demonstrates how to allow faculty logins on this system while refusing student logins.

To prevent specified users from logging on to a system, even if they are present in the NIS database, use `vipw` to add `-username` with the correct number of colons towards the end of /etc/master.passwd on the client, where *username* is the username of a user to bar from logging in. The line with the blocked user must be before the `+` line that allows NIS users. In this example, `bill` is barred from logging on to `basie`:

```
basie# cat /etc/master.passwd
root:[password]:0:0::0:0:The super-user:/root:/bin/csh
toor:[password]:0:0::0:0:The other super-user:/root:/bin/sh
daemon:*:1:1::0:0:Owner of many system processes:/root:/usr/sbin/nologin
operator:*:2:5::0:0:System &:/:/usr/sbin/nologin
bin:*:3:7::0:0:Binaries Commands and Source,,,:/:/usr/sbin/nologin
tty:*:4:65533::0:0:Tty Sandbox:/:/usr/sbin/nologin
kmem:*:5:65533::0:0:KMem Sandbox:/:/usr/sbin/nologin
games:*:7:13::0:0:Games pseudo-user:/usr/games:/usr/sbin/nologin
news:*:8:8::0:0:News Subsystem:/:/usr/sbin/nologin
man:*:9:9::0:0:Mister Man Pages:/usr/share/man:/usr/sbin/nologin
bind:*:53:53::0:0:Bind Sandbox:/:/usr/sbin/nologin
uucp:*:66:66::0:0:UUCP pseudo-user:/var/spool/uucppublic:/usr/libexec/uucp/uucico
xten:*:67:67::0:0:X-10 daemon:/usr/local/xten:/usr/sbin/nologin
pop:*:68:6::0:0:Post Office Owner:/nonexistent:/usr/sbin/nologin
nobody:*:65534:65534::0:0:Unprivileged user:/nonexistent:/usr/sbin/nologin
-bill:::::::::
+:::::::::

basie#
```

33.4.8. Using Netgroups

Barring specified users from logging on to individual systems becomes unscaleable on larger networks and quickly loses the main benefit of NIS: *centralized* administration.

Netgroups were developed to handle large, complex networks with hundreds of users and machines. Their use is comparable to UNIX® groups, where the main difference is the lack of a numeric ID and the ability to define a netgroup by including both user accounts and other netgroups.

To expand on the example used in this chapter, the NIS domain will be extended to add the users and systems shown in Tables 28.2 and 28.3:

Table 46. Additional Users

User Name(s)	Description
`alpha`, `beta`	IT department employees
`charlie`, `delta`	IT department apprentices
`echo`, `foxtrott`, `golf`, …	employees
`able`, `baker`, …	interns

Table 47. Additional Systems

Machine Name(s)	Description
`war`, `death`, `famine`, `pollution`	Only IT employees are allowed to log onto these servers.
`pride`, `greed`, `envy`, `wrath`, `lust`, `sloth`	All members of the IT department are allowed to login onto these servers.
`one`, `two`, `three`, `four`, …	Ordinary workstations used by employees.
`trashcan`	A very old machine without any critical data. Even interns are allowed to use this system.

When using netgroups to configure this scenario, each user is assigned to one or more netgroups and logins are then allowed or forbidden for all members of the netgroup. When adding a new machine, login restrictions must be defined for all netgroups. When a new user is added, the account must be added to one or more netgroups. If the NIS setup is planned carefully, only one central configuration file needs modification to grant or deny access to machines.

The first step is the initialization of the NIS `netgroup` map. In FreeBSD, this map is not created by default. On the NIS master server, use an editor to create a map named /var/yp/netgroup.

This example creates four netgroups to represent IT employees, IT apprentices, employees, and interns:

```
IT_EMP  (,alpha,test-domain)    (,beta,test-domain)
IT_APP  (,charlie,test-domain)  (,delta,test-domain)
USERS   (,echo,test-domain)     (,foxtrott,test-domain) \
        (,golf,test-domain)
INTERNS (,able,test-domain)     (,baker,test-domain)
```

Each entry configures a netgroup. The first column in an entry is the name of the netgroup. Each set of parentheses represents either a group of one or more users or the name of another netgroup. When specifying a user, the three comma-delimited fields inside each group represent:

1. The name of the host(s) where the other fields representing the user are valid. If a hostname is not specified, the entry is valid on all hosts.
2. The name of the account that belongs to this netgroup.
3. The NIS domain for the account. Accounts may be imported from other NIS domains into a netgroup.

If a group contains multiple users, separate each user with whitespace. Additionally, each field may contain wildcards. See netgroup(5) for details.

Netgroup names longer than 8 characters should not be used. The names are case sensitive and using capital letters for netgroup names is an easy way to distinguish between user, machine and netgroup names.

Some non-FreeBSD NIS clients cannot handle netgroups containing more than 15 entries. This limit may be circumvented by creating several sub-netgroups with 15 users or fewer and a real netgroup consisting of the sub-netgroups, as seen in this example:

```
BIGGRP1  (,joe1,domain)  (,joe2,domain)  (,joe3,domain) [...]
BIGGRP2  (,joe16,domain)  (,joe17,domain) [...]
BIGGRP3  (,joe31,domain)  (,joe32,domain)
BIGGROUP  BIGGRP1 BIGGRP2 BIGGRP3
```

Repeat this process if more than 225 (15 times 15) users exist within a single netgroup.

To activate and distribute the new NIS map:

```
ellington# cd /var/yp
ellington# make
```

This will generate the three NIS maps netgroup, netgroup.byhost and netgroup.byuser. Use the map key option of ypcat(1) to check if the new NIS maps are available:

```
ellington% ypcat -k netgroup
ellington% ypcat -k netgroup.byhost
ellington% ypcat -k netgroup.byuser
```

The output of the first command should resemble the contents of /var/yp/netgroup. The second command only produces output if host-specific netgroups were created. The third command is used to get the list of netgroups for a user.

To configure a client, use vipw(8) to specify the name of the netgroup. For example, on the server named `war`, replace this line:

```
+:::::::::
```

with

```
+@IT_EMP:::::::::
```

This specifies that only the users defined in the netgroup `IT_EMP` will be imported into this system's password database and only those users are allowed to login to this system.

This configuration also applies to the `~` function of the shell and all routines which convert between user names and numerical user IDs. In other words, `cd ~user` will not work, `ls -l` will show the numerical ID instead of the username, and `find . -user joe -print` will fail with the message `No such user`. To fix this, import all user entries without allowing them to login into the servers. This can be achieved by adding an extra line:

```
+:::::::::/usr/sbin/nologin
```

This line configures the client to import all entries but to replace the shell in those entries with /usr/sbin/nologin.

Make sure that extra line is placed *after* `+@IT_EMP:::::::::`. Otherwise, all user accounts imported from NIS will have /usr/sbin/nologin as their login shell and no one will be able to login to the system.

To configure the less important servers, replace the old `+:::::::::` on the servers with these lines:

```
+@IT_EMP:::::::::
+@IT_APP:::::::::
+:::::::::/usr/sbin/nologin
```

The corresponding lines for the workstations would be:

```
+@IT_EMP:::::::::
+@USERS:::::::::
+:::::::::/usr/sbin/nologin
```

NIS supports the creation of netgroups from other netgroups which can be useful if the policy regarding user access changes. One possibility is the creation of role-based netgroups. For example, one might create a netgroup called `BIGSRV` to define the login restrictions for the important servers, another netgroup called `SMALLSRV` for the less important servers, and a third netgroup called `USERBOX` for the workstations. Each of these netgroups contains the netgroups that are allowed to login onto these machines. The new entries for the NIS`netgroup` map would look like this:

```
BIGSRV    IT_EMP  IT_APP
SMALLSRV  IT_EMP  IT_APP  ITINTERN
USERBOX   IT_EMP  ITINTERN USERS
```

This method of defining login restrictions works reasonably well when it is possible to define groups of machines with identical restrictions. Unfortunately, this is the exception and not the rule. Most of the time, the ability to define login restrictions on a per-machine basis is required.

Machine-specific netgroup definitions are another possibility to deal with the policy changes. In this scenario, the /etc/master.passwd of each system contains two lines starting with "+". The first line adds a netgroup with the accounts allowed to login onto this machine and the second line adds all other accounts with /usr/sbin/nologin as shell. It is recommended to use the "ALL-CAPS" version of the hostname as the name of the netgroup:

```
+@BOXNAME:::::::::
+:::::::::/usr/sbin/nologin
```

Once this task is completed on all the machines, there is no longer a need to modify the local versions of /etc/master.passwd ever again. All further changes can be handled by modifying the NIS map. Here is an example of a possible `netgroup` map for this scenario:

```
# Define groups of users first
IT_EMP    (,alpha,test-domain)    (,beta,test-domain)
IT_APP    (,charlie,test-domain)  (,delta,test-domain)
DEPT1     (,echo,test-domain)     (,foxtrott,test-domain)
DEPT2     (,golf,test-domain)     (,hotel,test-domain)
DEPT3     (,india,test-domain)    (,juliet,test-domain)
ITINTERN  (,kilo,test-domain)     (,lima,test-domain)
D_INTERNS (,able,test-domain)     (,baker,test-domain)
#
# Now, define some groups based on roles
USERS     DEPT1   DEPT2     DEPT3
BIGSRV    IT_EMP  IT_APP
SMALLSRV  IT_EMP  IT_APP    ITINTERN
USERBOX   IT_EMP  ITINTERN  USERS
#
# And a groups for a special tasks
# Allow echo and golf to access our anti-virus-machine
SECURITY  IT_EMP  (,echo,test-domain)  (,golf,test-domain)
#
# machine-based netgroups
# Our main servers
WAR       BIGSRV
FAMINE    BIGSRV
# User india needs access to this server
POLLUTION  BIGSRV  (,india,test-domain)
#
# This one is really important and needs more access restrictions
DEATH     IT_EMP
#
# The anti-virus-machine mentioned above
ONE       SECURITY
#
# Restrict a machine to a single user
TWO       (,hotel,test-domain)
# [...more groups to follow]
```

It may not always be advisable to use machine-based netgroups. When deploying a couple of dozen or hundreds of systems, role-based netgroups instead of machine-based netgroups may be used to keep the size of the NIS map within reasonable limits.

33.4.9. Password Formats

NIS requires that all hosts within an NIS domain use the same format for encrypting passwords. If users have trouble authenticating on an NIS client, it may be due to a differing password format. In a heterogeneous network, the format must be supported by all operating systems, where DES is the lowest common standard.

To check which format a server or client is using, look at this section of /etc/login.conf:

```
default:\
    :passwd_format=des:\
    [Further entries elided]
```

In this example, the system is using the DES format for password hashing. Other possible values include `blf` for Blowfish, `md5` for MD5, `sha256` and `sha512` for SHA-256 and SHA-512 respectively. For more information and the up to date list of what is available on the system, consult the crypt(3) manpage.

If the format on a host needs to be edited to match the one being used in the NIS domain, the login capability database must be rebuilt after saving the change:

```
# cap_mkdb /etc/login.conf
```

The format of passwords for existing user accounts will not be updated until each user changes their password *after* the login capability database is rebuilt.

33.5. Lightweight Directory Access Protocol (LDAP)

The Lightweight Directory Access Protocol (LDAP) is an application layer protocol used to access, modify, and authenticate objects using a distributed directory information service. Think of it as a phone or record book which stores several levels of hierarchical, homogeneous information. It is used in Active Directory and OpenLDAP networks and allows users to access to several levels of internal information utilizing a single account. For example, email authentication, pulling employee contact information, and internal website authentication might all make use of a single user account in the LDAP server's record base.

This section provides a quick start guide for configuring an LDAP server on a FreeBSD system. It assumes that the administrator already has a design plan which includes the type of information to store, what that information will be used for, which users should have access to that information, and how to secure this information from unauthorized access.

33.5.1. LDAP Terminology and Structure

LDAP uses several terms which should be understood before starting the configuration. All directory entries consist of a group of *attributes*. Each of these attribute sets contains a unique identifier known as a *Distinguished Name* (DN) which is normally built from several other attributes such as the common or *Relative Distinguished Name* (RDN). Similar to how directories have absolute and relative paths, consider a DN as an absolute path and the RDN as the relative path.

An example LDAP entry looks like the following. This example searches for the entry for the specified user account (`uid`), organizational unit (`ou`), and organization (`o`):

```
% ldapsearch -xb "uid=trhodes,ou=users,o=example.com"
# extended LDIF
#
# LDAPv3
# base <uid=trhodes,ou=users,o=example.com> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# trhodes, users, example.com
dn: uid=trhodes,ou=users,o=example.com
mail: trhodes@example.com
cn: Tom Rhodes
uid: trhodes
telephoneNumber: (123) 456-7890

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

This example entry shows the values for the `dn`, `mail`, `cn`, `uid`, and `telephoneNumber` attributes. The cn attribute is the RDN.

More information about LDAP and its terminology can be found at http://www.openldap.org/doc/admin24/intro.html.

33.5.2. Configuring an LDAP Server

FreeBSD does not provide a built-in LDAP server. Begin the configuration by installing net/openldap-server package or port:

```
# pkg install openldap-server
```

There is a large set of default options enabled in the package. Review them by running `pkg info openldap-server`. If they are not sufficient (for example if SQL support is needed), please consider recompiling the port using the appropriate framework.

The installation creates the directory /var/db/openldap-data to hold the data. The directory to store the certificates must be created:

```
# mkdir /usr/local/etc/openldap/private
```

The next phase is to configure the Certificate Authority. The following commands must be executed from /usr/local/etc/openldap/private. This is important as the file permissions need to be restrictive and users should not have access to these files. More detailed information about certificates and their parameters can be found in OpenSSL. To create the Certificate Authority, start with this command and follow the prompts:

```
# openssl req -days 365 -nodes -new -x509 -keyout ca.key -out ../ca.crt
```

The entries for the prompts may be generic *except* for the `Common Name`. This entry must be *different* than the system hostname. If this will be a self signed certificate, prefix the hostname with `CA` for Certificate Authority.

The next task is to create a certificate signing request and a private key. Input this command and follow the prompts:

```
# openssl req -days 365 -nodes -new -keyout server.key -out server.csr
```

During the certificate generation process, be sure to correctly set the `Common Name` attribute. The Certificate Signing Request must be signed with the Certificate Authority in order to be used as a valid certificate:

```
# openssl x509 -req -days 365 -in server.csr -out ../server.crt -CA ../ca.crt -CAkey
ca.key -CAcreateserial
```

The final part of the certificate generation process is to generate and sign the client certificates:

```
# openssl req -days 365 -nodes -new -keyout client.key -out client.csr
# openssl x509 -req -days 3650 -in client.csr -out ../client.crt -CA ../ca.crt -CAkey
ca.key
```

Remember to use the same `Common Name` attribute when prompted. When finished, ensure that a total of eight (8) new files have been generated through the proceeding commands.

The daemon running the OpenLDAP server is slapd. Its configuration is performed through slapd.ldif: the old slapd.conf has been deprecated by OpenLDAP.

Configuration examples for slapd.ldif are available and can also be found in /usr/local/etc/openldap/slapd.ldif.sample. Options are documented in slapd-config(5). Each section of slapd.ldif, like all the other LDAP attribute sets, is uniquely identified through a DN. Be sure that no blank lines are left between the `dn:` statement and the desired end of the section. In the following example, TLS will be used to implement a secure channel. The first section represents the global configuration:

```
#
# See slapd-config(5) for details on configuration options.
# This file should NOT be world readable.
#
dn: cn=config
objectClass: olcGlobal
cn: config
#
#
# Define global ACLs to disable default read access.
#
olcArgsFile: /var/run/openldap/slapd.args
olcPidFile: /var/run/openldap/slapd.pid
olcTLSCertificateFile: /usr/local/etc/openldap/server.crt
olcTLSCertificateKeyFile: /usr/local/etc/openldap/private/server.key
olcTLSCACertificateFile: /usr/local/etc/openldap/ca.crt
#olcTLSCipherSuite: HIGH
olcTLSProtocolMin: 3.1
olcTLSVerifyClient: never
```

The Certificate Authority, server certificate and server private key files must be specified here. It is recommended to let the clients choose the security cipher and omit option `olcTLSCipherSuite` (incompatible with TLS clients other than openssl). Option `olcTLSProtocolMin` lets the server require a minimum security level: it is recommended. While verification is mandatory for the server, it is not for the client: `olcTLSVerifyClient: never`.

The second section is about the backend modules and can be configured as follows:

```
#
# Load dynamic backend modules:
#
dn: cn=module,cn=config
objectClass: olcModuleList
cn: module
olcModulepath:  /usr/local/libexec/openldap
olcModuleload:  back_mdb.la
#olcModuleload: back_bdb.la
#olcModuleload: back_hdb.la
#olcModuleload: back_ldap.la
#olcModuleload: back_passwd.la
#olcModuleload: back_shell.la
```

The third section is devoted to load the needed `ldif` schemas to be used by the databases: they are essential.

```
dn: cn=schema,cn=config
objectClass: olcSchemaConfig
cn: schema

include: file:///usr/local/etc/openldap/schema/core.ldif
include: file:///usr/local/etc/openldap/schema/cosine.ldif
include: file:///usr/local/etc/openldap/schema/inetorgperson.ldif
include: file:///usr/local/etc/openldap/schema/nis.ldif
```

Next, the frontend configuration section:

```
# Frontend settings
#
dn: olcDatabase={-1}frontend,cn=config
objectClass: olcDatabaseConfig
objectClass: olcFrontendConfig
olcDatabase: {-1}frontend
olcAccess: to * by * read
#
# Sample global access control policy:
#   Root DSE: allow anyone to read it
#   Subschema (sub)entry DSE: allow anyone to read it
#   Other DSEs:
#       Allow self write access
#       Allow authenticated users read access
#       Allow anonymous users to authenticate
#
#olcAccess: to dn.base="" by * read
#olcAccess: to dn.base="cn=Subschema" by * read
#olcAccess: to *
#   by self write
#   by users read
#   by anonymous auth
#
# if no access controls are present, the default policy
# allows anyone and everyone to read anything but restricts
# updates to rootdn.  (e.g., "access to * by * read")
#
# rootdn can always read and write EVERYTHING!
#
olcPasswordHash: {SSHA}
# {SSHA} is already the default for olcPasswordHash
```

Another section is devoted to the *configuration backend*, the only way to later access the OpenLDAP server configuration is as a global super-user.

```
dn: olcDatabase={0}config,cn=config
objectClass: olcDatabaseConfig
```

```
olcDatabase: {0}config
olcAccess: to * by * none
olcRootPW: {SSHA}iae+lrQZILpiUdf16Z9KmDmSwT77Dj4U
```

The default administrator username is `cn=config`. Type slappasswd in a shell, choose a password and use its hash in `olcRootPW`. If this option is not specified now, before slapd.ldif is imported, no one will be later able to modify the *global configuration* section.

The last section is about the database backend:

```
#######################################################################
# LMDB database definitions
#######################################################################
#
dn: olcDatabase=mdb,cn=config
objectClass: olcDatabaseConfig
objectClass: olcMdbConfig
olcDatabase: mdb
olcDbMaxSize: 1073741824
olcSuffix: dc=domain,dc=example
olcRootDN: cn=mdbadmin,dc=domain,dc=example
# Cleartext passwords, especially for the rootdn, should
# be avoided.  See slappasswd(8) and slapd-config(5) for details.
# Use of strong authentication encouraged.
olcRootPW: {SSHA}X2wHvIWDk6G76CQyCMS1vDCvtICWgn0+
# The database directory MUST exist prior to running slapd AND
# should only be accessible by the slapd and slap tools.
# Mode 700 recommended.
olcDbDirectory: /var/db/openldap-data
# Indices to maintain
olcDbIndex: objectClass eq
```

This database hosts the *actual contents* of the LDAP directory. Types other than `mdb` are available. Its super-user, not to be confused with the global one, is configured here: a (possibly custom) username in `olcRootDN` and the password hash in `olcRootPW`; slappasswd can be used as before.

This repository contains four examples of slapd.ldif. To convert an existing slapd.conf into slapd.ldif, refer to this page (please note that this may introduce some unuseful options).

When the configuration is completed, slapd.ldif must be placed in an empty directory. It is recommended to create it as:

```
# mkdir /usr/local/etc/openldap/slapd.d/
```

Import the configuration database:

```
# /usr/local/sbin/slapadd -n0 -F /usr/local/etc/openldap/slapd.d/ -l
```

```
/usr/local/etc/openldap/slapd.ldif
```

Start the slapd daemon:

```
# /usr/local/libexec/slapd -F /usr/local/etc/openldap/slapd.d/
```

Option `-d` can be used for debugging, as specified in slapd(8). To verify that the server is running and working:

```
# ldapsearch -x -b '' -s base '(objectclass=*)' namingContexts
# extended LDIF
#
# LDAPv3
# base <> with scope baseObject
# filter: (objectclass=*)
# requesting: namingContexts
#

#
dn:
namingContexts: dc=domain,dc=example

# search result
search: 2
result: 0 Success

# numResponses: 2
# numEntries: 1
```

The server must still be trusted. If that has never been done before, follow these instructions. Install the OpenSSL package or port:

```
# pkg install openssl
```

From the directory where ca.crt is stored (in this example, /usr/local/etc/openldap), run:

```
# c_rehash .
```

Both the CA and the server certificate are now correctly recognized in their respective roles. To verify this, run this command from the server.crt directory:

```
# openssl verify -verbose -CApath . server.crt
```

If slapd was running, restart it. As stated in /usr/local/etc/rc.d/slapd, to properly run slapd at boot the following lines must be added to /etc/rc.conf:

```
slapd_enable="YES"
slapd_flags='-h "ldapi://%2fvar%2frun%2fopenldap%2fldapi/ ldap://0.0.0.0/"'
slapd_sockets="/var/run/openldap/ldapi"
slapd_cn_config="YES"
```

slapd does not provide debugging at boot. Check /var/log/debug.log, dmesg -a and /var/log/messages for this purpose.

The following example adds the group `team` and the user `john` to the `domain.example` LDAP database, which is still empty. First, create the file domain.ldif:

```
# cat domain.ldif
dn: dc=domain,dc=example
objectClass: dcObject
objectClass: organization
o: domain.example
dc: domain

dn: ou=groups,dc=domain,dc=example
objectClass: top
objectClass: organizationalunit
ou: groups

dn: ou=users,dc=domain,dc=example
objectClass: top
objectClass: organizationalunit
ou: users

dn: cn=team,ou=groups,dc=domain,dc=example
objectClass: top
objectClass: posixGroup
cn: team
gidNumber: 10001

dn: uid=john,ou=users,dc=domain,dc=example
objectClass: top
objectClass: account
objectClass: posixAccount
objectClass: shadowAccount
cn: John McUser
uid: john
uidNumber: 10001
gidNumber: 10001
homeDirectory: /home/john/
loginShell: /usr/bin/bash
userPassword: secret
```

See the OpenLDAP documentation for more details. Use slappasswd to replace the plain text password `secret` with a hash in `userPassword`. The path specified as `loginShell` must exist in all the systems where `john` is allowed to login. Finally, use the `mdb` administrator to modify the database:

```
# ldapadd -W -D "cn=mdbadmin,dc=domain,dc=example" -f domain.ldif
```

Modifications to the *global configuration* section can only be performed by the global super-user. For example, assume that the option `olcTLSCipherSuite: HIGH:MEDIUM:SSLv3` was initially specified and must now be deleted. First, create a file that contains the following:

```
# cat global_mod
dn: cn=config
changetype: modify
delete: olcTLSCipherSuite
```

Then, apply the modifications:

```
# ldapmodify -f global_mod -x -D "cn=config" -W
```

When asked, provide the password chosen in the *configuration backend* section. The username is not required: here, `cn=config` represents the DN of the database section to be modified. Alternatively, use `ldapmodify` to delete a single line of the database, `ldapdelete` to delete a whole entry.

If something goes wrong, or if the global super-user cannot access the configuration backend, it is possible to delete and re-write the whole configuration:

```
# rm -rf /usr/local/etc/openldap/slapd.d/
```

slapd.ldif can then be edited and imported again. Please, follow this procedure only when no other solution is available.

This is the configuration of the server only. The same machine can also host an LDAP client, with its own separate configuration.

33.6. Dynamic Host Configuration Protocol (DHCP)

The Dynamic Host Configuration Protocol (DHCP) allows a system to connect to a network in order to be assigned the necessary addressing information for communication on that network. FreeBSD includes the OpenBSD version of `dhclient` which is used by the client to obtain the addressing information. FreeBSD does not install a DHCP server, but several servers are available in the FreeBSD Ports Collection. The DHCP protocol is fully described in RFC 2131. Informational resources are also available at isc.org/downloads/dhcp/.

This section describes how to use the built-in DHCP client. It then describes how to install and configure a DHCP server.

In FreeBSD, the bpf(4) device is needed by both the DHCP server and DHCP client. This device is included in the GENERIC kernel that is installed with FreeBSD. Users who prefer to create a custom kernel need to keep this device if DHCP is used.

It should be noted that bpf also allows privileged users to run network packet sniffers on that system.

33.6.1. Configuring a DHCP Client

DHCP client support is included in the FreeBSD installer, making it easy to configure a newly installed system to automatically receive its networking addressing information from an existing DHCP server. Refer to Accounts, Time Zone, Services and Hardening for examples of network configuration.

When `dhclient` is executed on the client machine, it begins broadcasting requests for configuration information. By default, these requests use UDP port 68. The server replies on UDP port 67, giving the client an IP address and other relevant network information such as a subnet mask, default gateway, and DNS server addresses. This information is in the form of a DHCP "lease" and is valid for a configurable time. This allows stale IP addresses for clients no longer connected to the network to automatically be reused. DHCP clients can obtain a great deal of information from the server. An exhaustive list may be found in dhcp-options(5).

By default, when a FreeBSD system boots, its DHCP client runs in the background, or *asynchronously*. Other startup scripts continue to run while the DHCP process completes, which speeds up system startup.

Background DHCP works well when the DHCP server responds quickly to the client's requests. However, DHCP may take a long time to complete on some systems. If network services attempt to run before DHCP has assigned the network addressing information, they will fail. Using DHCP in *synchronous* mode prevents this problem as it pauses startup until the DHCP configuration has completed.

This line in /etc/rc.conf is used to configure background or asynchronous mode:

```
ifconfig_fxp0="DHCP"
```

This line may already exist if the system was configured to use DHCP during installation. Replace the *fxp0* shown in these examples with the name of the interface to be dynamically configured, as described in "Setting Up Network Interface Cards".

To instead configure the system to use synchronous mode, and to pause during startup while DHCP completes, use "SYNCDHCP":

```
ifconfig_fxp0="SYNCDHCP"
```

Additional client options are available. Search for `dhclient` in rc.conf(5) for details.

The DHCP client uses the following files:

- /etc/dhclient.conf

  The configuration file used by `dhclient`. Typically, this file contains only comments as the defaults are suitable for most clients. This configuration file is described in dhclient.conf(5).

- /sbin/dhclient

  More information about the command itself can be found in dhclient(8).

- /sbin/dhclient-script

  The FreeBSD-specific DHCP client configuration script. It is described in dhclient-script(8), but should not need any user modification to function properly.

- /var/db/dhclient.leases.interface

  The DHCP client keeps a database of valid leases in this file, which is written as a log and is described in dhclient.leases(5).

33.6.2. Installing and Configuring a DHCP Server

This section demonstrates how to configure a FreeBSD system to act as a DHCP server using the Internet Systems Consortium (ISC) implementation of the DHCP server. This implementation and its documentation can be installed using the net/isc-dhcp44-server package or port.

The installation of net/isc-dhcp44-server installs a sample configuration file. Copy /usr/local/etc/dhcpd.conf.example to /usr/local/etc/dhcpd.conf and make any edits to this new file.

The configuration file is comprised of declarations for subnets and hosts which define the information that is provided to DHCP clients. For example, these lines configure the following:

```
option domain-name "example.org";①
option domain-name-servers ns1.example.org;②
option subnet-mask 255.255.255.0;③

default-lease-time 600;④
max-lease-time 72400;⑤
ddns-update-style none;⑥

subnet 10.254.239.0 netmask 255.255.255.224 {
  range 10.254.239.10 10.254.239.20;⑦
  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;⑧
}

host fantasia {
  hardware ethernet 08:00:07:26:c0:a5;⑨
  fixed-address fantasia.fugue.com;⑩
}
```

① This option specifies the default search domain that will be provided to clients. Refer to resolv.conf(5) for more information.

② This option specifies a comma separated list of DNS servers that the client should use. They can be listed by their Fully Qualified Domain Names (FQDN), as seen in the example, or by their IP addresses.

③ The subnet mask that will be provided to clients.

④ The default lease expiry time in seconds. A client can be configured to override this value.

⑤ The maximum allowed length of time, in seconds, for a lease. Should a client request a longer lease, a lease will still be issued, but it will only be valid for `max-lease-time`.

⑥ The default of `none` disables dynamic DNS updates. Changing this to `interim` configures the DHCP server to update a DNS server whenever it hands out a lease so that the DNS server knows which IP addresses are associated with which computers in the network. Do not change the default setting unless the DNS server has been configured to support dynamic DNS.

⑦ This line creates a pool of available IP addresses which are reserved for allocation to DHCP clients. The range of addresses must be valid for the network or subnet specified in the previous line.

⑧ Declares the default gateway that is valid for the network or subnet specified before the opening `{` bracket.

⑨ Specifies the hardware MAC address of a client so that the DHCP server can recognize the client when it makes a request.

⑩ Specifies that this host should always be given the same IP address. Using the hostname is correct, since the DHCP server will resolve the hostname before returning the lease information.

This configuration file supports many more options. Refer to dhcpd.conf(5), installed with the server, for details and examples.

Once the configuration of dhcpd.conf is complete, enable the DHCP server in /etc/rc.conf:

```
dhcpd_enable="YES"
dhcpd_ifaces="dc0"
```

Replace the `dc0` with the interface (or interfaces, separated by whitespace) that the DHCP server should listen on for DHCP client requests.

Start the server by issuing the following command:

```
# service isc-dhcpd start
```

Any future changes to the configuration of the server will require the dhcpd service to be stopped and then started using service(8).

The DHCP server uses the following files. Note that the manual pages are installed with the server software.

- /usr/local/sbin/dhcpd

  More information about the dhcpd server can be found in dhcpd(8).

- /usr/local/etc/dhcpd.conf

  The server configuration file needs to contain all the information that should be provided to clients, along with information regarding the operation of the server. This configuration file is described in dhcpd.conf(5).

- /var/db/dhcpd.leases

  The DHCP server keeps a database of leases it has issued in this file, which is written as a log. Refer to dhcpd.leases(5), which gives a slightly longer description.

- /usr/local/sbin/dhcrelay

  This daemon is used in advanced environments where one DHCP server forwards a request from a client to another DHCP server on a separate network. If this functionality is required, install the net/isc-dhcp44-relay package or port. The installation includes dhcrelay(8) which provides more detail.

33.7. Domain Name System (DNS)

Domain Name System (DNS) is the protocol through which domain names are mapped to IP addresses, and vice versa. DNS is coordinated across the Internet through a somewhat complex system of authoritative root, Top Level Domain (TLD), and other smaller-scale name servers, which host and cache individual domain information. It is not necessary to run a name server to perform DNS lookups on a system.

The following table describes some of the terms associated with DNS:

Table 48. DNS Terminology

Term	Definition
Forward DNS	Mapping of hostnames to IP addresses.
Origin	Refers to the domain covered in a particular zone file.
Resolver	A system process through which a machine queries a name server for zone information.
Reverse DNS	Mapping of IP addresses to hostnames.
Root zone	The beginning of the Internet zone hierarchy. All zones fall under the root zone, similar to how all files in a file system fall under the root directory.

Term	Definition
Zone	An individual domain, subdomain, or portion of the DNS administered by the same authority.

Examples of zones:

- `.` is how the root zone is usually referred to in documentation.
- `org.` is a Top Level Domain (TLD) under the root zone.
- `example.org.` is a zone under the `org.`TLD.
- `1.168.192.in-addr.arpa` is a zone referencing all IP addresses which fall under the `192.168.1.*`IP address space.

As one can see, the more specific part of a hostname appears to its left. For example, `example.org.` is more specific than `org.`, as `org.` is more specific than the root zone. The layout of each part of a hostname is much like a file system: the /dev directory falls within the root, and so on.

33.7.1. Reasons to Run a Name Server

Name servers generally come in two forms: authoritative name servers, and caching (also known as resolving) name servers.

An authoritative name server is needed when:

- One wants to serve DNS information to the world, replying authoritatively to queries.
- A domain, such as `example.org`, is registered and IP addresses need to be assigned to hostnames under it.
- An IP address block requires reverse DNS entries (IP to hostname).
- A backup or second name server, called a slave, will reply to queries.

A caching name server is needed when:

- A local DNS server may cache and respond more quickly than querying an outside name server.

When one queries for `www.FreeBSD.org`, the resolver usually queries the uplink ISP's name server, and retrieves the reply. With a local, caching DNS server, the query only has to be made once to the outside world by the caching DNS server. Additional queries will not have to go outside the local network, since the information is cached locally.

33.7.2. DNS Server Configuration

Unbound is provided in the FreeBSD base system. By default, it will provide DNS resolution to the local machine only. While the base system package can be configured to provide resolution services beyond the local machine, it is recommended that such requirements be addressed by installing Unbound from the FreeBSD Ports Collection.

To enable Unbound, add the following to /etc/rc.conf:

```
local_unbound_enable="YES"
```

Any existing nameservers in /etc/resolv.conf will be configured as forwarders in the new Unbound configuration.

> If any of the listed nameservers do not support DNSSEC, local DNS resolution will fail. Be sure to test each nameserver and remove any that fail the test. The following command will show the trust tree or a failure for a nameserver running on `192.168.1.1`:
>
> ```
> % drill -S FreeBSD.org @192.168.1.1
> ```

Once each nameserver is confirmed to support DNSSEC, start Unbound:

```
# service local_unbound onestart
```

This will take care of updating /etc/resolv.conf so that queries for DNSSEC secured domains will now work. For example, run the following to validate the FreeBSD.org DNSSEC trust tree:

```
% drill -S FreeBSD.org
;; Number of trusted keys: 1
;; Chasing: freebsd.org. A

DNSSEC Trust tree:
freebsd.org. (A)
|---freebsd.org. (DNSKEY keytag: 36786 alg: 8 flags: 256)
    |---freebsd.org. (DNSKEY keytag: 32659 alg: 8 flags: 257)
    |---freebsd.org. (DS keytag: 32659 digest type: 2)
        |---org. (DNSKEY keytag: 49587 alg: 7 flags: 256)
            |---org. (DNSKEY keytag: 9795 alg: 7 flags: 257)
            |---org. (DNSKEY keytag: 21366 alg: 7 flags: 257)
            |---org. (DS keytag: 21366 digest type: 1)
            |   |---. (DNSKEY keytag: 40926 alg: 8 flags: 256)
            |       |---. (DNSKEY keytag: 19036 alg: 8 flags: 257)
            |---org. (DS keytag: 21366 digest type: 2)
                |---. (DNSKEY keytag: 40926 alg: 8 flags: 256)
                    |---. (DNSKEY keytag: 19036 alg: 8 flags: 257)
;; Chase successful
```

33.7.3. Authoritative Name Server Configuration

FreeBSD does not provide authoritative name server software in the base system. Users are encouraged to install third party applications, like dns/nsd or dns/bind918 package or port.

33.8. Zero-configuration Networking (mDNS/DNS-SD)

Zero-configuration networking (sometimes referred to as *Zeroconf*) is a set of technologies, which simplify network configuration. The main parts of Zeroconf are:

- Link-Local Addressing providing automatic assignment of numeric network addresses.
- Multicast DNS (*mDNS*) providing automatic distribution and resolution of hostnames.
- DNS-Based Service Discovery (*DNS-SD*) providing automatic discovery of service instances.

33.8.1. Configuring and Starting Avahi

One of the popular implementations of zeroconf is Avahi. Avahi can be installed and configured with the following commands:

```
# pkg install avahi-app nss_mdns
# grep -q '^hosts:.*\<mdns\>' /etc/nsswitch.conf || sed -i "" 's/^hosts: .*/& mdns/'
/etc/nsswitch.conf
# service dbus enable
# service avahi-daemon enable
# service dbus start
# service avahi-daemon start
```

33.9. Apache HTTP Server

The open source Apache HTTP Server is the most widely used web server. FreeBSD does not install this web server by default, but it can be installed from the www/apache24 package or port.

This section summarizes how to configure and start version 2.*x* of the Apache HTTP Server on FreeBSD. For more detailed information about Apache 2.X and its configuration directives, refer to httpd.apache.org.

33.9.1. Configuring and Starting Apache

In FreeBSD, the main Apache HTTP Server configuration file is installed as /usr/local/etc/apache2x/httpd.conf, where *x* represents the version number. This ASCII text file begins comment lines with a `#`. The most frequently modified directives are:

`ServerRoot "/usr/local"`

Specifies the default directory hierarchy for the Apache installation. Binaries are stored in the bin and sbin subdirectories of the server root and configuration files are stored in the etc/apache2x subdirectory.

`ServerAdmin you@example.com`

Change this to the email address to receive problems with the server. This address also appears on some server-generated pages, such as error documents.

`ServerName www.example.com:80`

Allows an administrator to set a hostname which is sent back to clients for the server. For example, `www` can be used instead of the actual hostname. If the system does not have a registered DNS name, enter its IP address instead. If the server will listen on an alternate report, change `80` to the alternate port number.

`DocumentRoot "/usr/local/www/apache2_x_/data"`

The directory where documents will be served from. By default, all requests are taken from this directory, but symbolic links and aliases may be used to point to other locations.

It is always a good idea to make a backup copy of the default Apache configuration file before making changes. When the configuration of Apache is complete, save the file and verify the configuration using `apachectl`. Running `apachectl configtest` should return `Syntax OK`.

To launch Apache at system startup, add the following line to /etc/rc.conf:

```
apache24_enable="YES"
```

If Apache should be started with non-default options, the following line may be added to /etc/rc.conf to specify the needed flags:

```
apache24_flags=""
```

If apachectl does not report configuration errors, start `httpd` now:

```
# service apache24 start
```

The `httpd` service can be tested by entering `http://localhost` in a web browser, replacing *localhost* with the fully-qualified domain name of the machine running `httpd`. The default web page that is displayed is /usr/local/www/apache24/data/index.html.

The Apache configuration can be tested for errors after making subsequent configuration changes while `httpd` is running using the following command:

```
# service apache24 configtest
```

> It is important to note that `configtest` is not an rc(8) standard, and should not be expected to work for all startup scripts.

33.9.2. Virtual Hosting

Virtual hosting allows multiple websites to run on one Apache server. The virtual hosts can be *IP-based* or *name-based*. IP-based virtual hosting uses a different IP address for each website. Name-based virtual hosting uses the clients HTTP/1.1 headers to figure out the hostname, which allows the websites to share the same IP address.

To setup Apache to use name-based virtual hosting, add a `VirtualHost` block for each website. For example, for the webserver named `www.domain.tld` with a virtual domain of `www.someotherdomain.tld`, add the following entries to httpd.conf:

```
<VirtualHost *>
    ServerName www.domain.tld
    DocumentRoot /www/domain.tld
</VirtualHost>

<VirtualHost *>
    ServerName www.someotherdomain.tld
    DocumentRoot /www/someotherdomain.tld
</VirtualHost>
```

For each virtual host, replace the values for `ServerName` and `DocumentRoot` with the values to be used.

For more information about setting up virtual hosts, consult the official Apache documentation at: http://httpd.apache.org/docs/vhosts/.

33.9.3. Apache Modules

Apache uses modules to augment the functionality provided by the basic server. Refer to http://httpd.apache.org/docs/current/mod/ for a complete listing of and the configuration details for the available modules.

In FreeBSD, some modules can be compiled with the www/apache24 port. Type `make config` within /usr/ports/www/apache24 to see which modules are available and which are enabled by default. If the module is not compiled with the port, the FreeBSD Ports Collection provides an easy way to install many modules. This section describes three of the most commonly used modules.

SSL support

At one point, support for SSL inside of Apache required a secondary module called mod_ssl. This is no longer the case and the default install of Apache comes with SSL built into the web server. An example of how to enable support for SSL websites is available in the installed file, httpd-ssl.conf inside of the /usr/local/etc/apache24/extra directory Inside this directory is also a sample file called named ssl.conf-sample. It is recommended that both files be evaluated to properly set up secure websites in the Apache web server.

After the configuration of SSL is complete, the following line must be uncommented in the main http.conf to activate the changes on the next restart or reload of Apache:

```
#Include etc/apache24/extra/httpd-ssl.conf
```

SSL version two and version three have known vulnerability issues. It is highly recommended TLS version 1.2 and 1.3 be enabled in place of the older SSL options.

> This can be accomplished by setting the following options in the ssl.conf:

```
SSLProtocol all -SSLv3 -SSLv2 +TLSv1.2 +TLSv1.3
SSLProxyProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
```

To complete the configuration of SSL in the web server, uncomment the following line to ensure that the configuration will be pulled into Apache during restart or reload:

```
# Secure (SSL/TLS) connections
Include etc/apache24/extra/httpd-ssl.conf
```

The following lines must also be uncommented in the httpd.conf to fully support SSL in Apache:

```
LoadModule authn_socache_module libexec/apache24/mod_authn_socache.so
LoadModule socache_shmcb_module libexec/apache24/mod_socache_shmcb.so
LoadModule ssl_module libexec/apache24/mod_ssl.so
```

The next step is to work with a certificate authority to have the appropriate certificates installed on the system. This will set up a chain of trust for the site and prevent any warnings of self-signed certificates.

mod_perl

The mod_perl module makes it possible to write Apache modules in Perl. In addition, the persistent interpreter embedded in the server avoids the overhead of starting an external interpreter and the penalty of Perl start-up time.

The mod_perl can be installed using the www/mod_perl2 package or port. Documentation for using this module can be found at http://perl.apache.org/docs/2.0/index.html.

mod_php

PHP: Hypertext Preprocessor (PHP) is a general-purpose scripting language that is especially suited for web development. Capable of being embedded into HTML, its syntax draws upon C, Java™, and Perl with the intention of allowing web developers to write dynamically generated webpages quickly.

Support for PHP for Apache and any other feature written in the language, can be added by installing the appropriate port.

For all supported versions, search the package database using `pkg`:

```
# pkg search php
```

A list will be displayed including the versions and additional features they provide. The components are completely modular, meaning features are enabled by installing the appropriate port. To install PHP version 7.4 for Apache, issue the following command:

```
# pkg install mod_php74
```

If any dependency packages need to be installed, they will be installed as well.

By default, PHP will not be enabled. The following lines will need to be added to the Apache configuration file located in /usr/local/etc/apache24 to make it active:

```
<FilesMatch "\.php$">
    SetHandler application/x-httpd-php
</FilesMatch>
<FilesMatch "\.phps$">
    SetHandler application/x-httpd-php-source
</FilesMatch>
```

In addition, the `DirectoryIndex` in the configuration file will also need to be updated and Apache will either need to be restarted or reloaded for the changes to take effect.

Support for many of the PHP features may also be installed by using `pkg`. For example, to install support for XML or SSL, install their respective ports:

```
# pkg install php74-xml php74-openssl
```

As before, the Apache configuration will need to be reloaded for the changes to take effect, even in cases where it was just a module install.

To perform a graceful restart to reload the configuration, issue the following command:

```
# apachectl graceful
```

Once the install is complete, there are two methods of obtaining the installed PHP support modules and the environmental information of the build. The first is to install the full PHP binary and running the command to gain the information:

```
# pkg install php74
```

```
# php -i | less
```

It is necessary to pass the output to a pager, such as the `more` or `less` to easier digest the amount of output.

Finally, to make any changes to the global configuration of PHP there is a well documented file installed into /usr/local/etc/php.ini. At the time of install, this file will not exist because there are two versions to choose from, one is php.ini-development and the other is php.ini-production. These are starting points to assist administrators in their deployment.

HTTP2 Support

Apache support for the HTTP2 protocol is included by default when installing the port with `pkg`. The new version of HTTP includes many improvements over the previous version, including utilizing a single connection to a website, reducing overall roundtrips of TCP connections. Also, packet header data is compressed and HTTP2 requires encryption by default.

When Apache is configured to only use HTTP2, web browsers will require secure, encrypted HTTPS connections. When Apache is configured to use both versions, HTTP1.1 will be considered a fall back option if any issues arise during the connection.

While this change does require administrators to make changes, they are positive and equate to a more secure Internet for everyone. The changes are only required for sites not currently implementing SSL and TLS.

> This configuration depends on the previous sections, including TLS support. It is recommended those instructions be followed before continuing with this configuration.

Start the process by enabling the http2 module by uncommenting the line in /usr/local/etc/apache24/httpd.conf and replace the mpm_prefork module with mpm_event as the former does not support HTTP2.

```
LoadModule http2_module libexec/apache24/mod_http2.so
LoadModule mpm_event_module libexec/apache24/mod_mpm_event.so
```

> There is a separate mod_http2 port that is available. It exists to deliver security and bug fixes quicker than the module installed with the bundled apache24 port. It is not required for HTTP2 support but is available. When installed, the mod_h2.so should be used in place of mod_http2.so in the Apache configuration.

There are two methods to implement HTTP2 in Apache; one way is globally for all sites and each VirtualHost running on the system. To enable HTTP2 globally, add the following line under the ServerName directive:

```
Protocols h2 http/1.1
```

> To enable HTTP2 over plaintext, use h2h2chttp/1.1 in the httpd.conf.

Having the h2c here will allow plaintext HTTP2 data to pass on the system but is not recommended. In addition, using the http/1.1 here will allow fallback to the HTTP1.1 version of the protocol should it be needed by the system.

To enable HTTP2 for individual VirtualHosts, add the same line within the VirtualHost directive in either httpd.conf or httpd-ssl.conf.

Reload the configuration using the `apachectl`reload command and test the configuration either by using either of the following methods after visiting one of the hosted pages:

```
# grep "HTTP/2.0" /var/log/httpd-access.log
```

This should return something similar to the following:

```
192.168.1.205 - - [18/Oct/2020:18:34:36 -0400] "GET / HTTP/2.0" 304 -
192.0.2.205 - - [18/Oct/2020:19:19:57 -0400] "GET / HTTP/2.0" 304 -
192.0.0.205 - - [18/Oct/2020:19:20:52 -0400] "GET / HTTP/2.0" 304 -
192.0.2.205 - - [18/Oct/2020:19:23:10 -0400] "GET / HTTP/2.0" 304 -
```

The other method is using the web browser's built in site debugger or `tcpdump`; however, using either method is beyond the scope of this document.

Support for HTTP2 reverse proxy connections by using the mod_proxy_http2.so module. When configuring the ProxyPass or RewriteRules [P] statements, they should use h2:// for the connection.

33.9.4. Dynamic Websites

In addition to mod_perl and mod_php, other languages are available for creating dynamic web content. These include Django and Ruby on Rails.

Django

Django is a BSD-licensed framework designed to allow developers to write high performance, elegant web applications quickly. It provides an object-relational mapper so that data types are developed as Python objects. A rich dynamic database-access API is provided for those objects without the developer ever having to write SQL. It also provides an extensible template system so that the logic of the application is separated from the HTML presentation.

Django depends on mod_python, and an SQL database engine. In FreeBSD, the www/py-django port automatically installs mod_python and supports the PostgreSQL, MySQL, or SQLite databases, with the default being SQLite. To change the database engine, type `make config` within /usr/ports/www/py-django, then install the port.

Once Django is installed, the application will need a project directory along with the Apache configuration in order to use the embedded Python interpreter. This interpreter is used to call the application for specific URLs on the site.

To configure Apache to pass requests for certain URLs to the web application, add the following to httpd.conf, specifying the full path to the project directory:

```
<Location "/">
```

```
    SetHandler python-program
    PythonPath "['/dir/to/the/django/packages/'] + sys.path"
    PythonHandler django.core.handlers.modpython
    SetEnv DJANGO_SETTINGS_MODULE mysite.settings
    PythonAutoReload On
    PythonDebug On
</Location>
```

Refer to https://docs.djangoproject.com for more information on how to use Django.

Ruby on Rails

Ruby on Rails is another open source web framework that provides a full development stack. It is optimized to make web developers more productive and capable of writing powerful applications quickly. On FreeBSD, it can be installed using the www/rubygem-rails package or port.

Refer to http://guides.rubyonrails.org for more information on how to use Ruby on Rails.

33.10. File Transfer Protocol (FTP)

The File Transfer Protocol (FTP) provides users with a simple way to transfer files to and from an FTP server. FreeBSD includes FTP server software, ftpd, in the base system.

FreeBSD provides several configuration files for controlling access to the FTP server. This section summarizes these files. Refer to ftpd(8) for more details about the built-in FTP server.

33.10.1. Configuration

The most important configuration step is deciding which accounts will be allowed access to the FTP server. A FreeBSD system has a number of system accounts which should not be allowed FTP access. The list of users disallowed any FTP access can be found in /etc/ftpusers. By default, it includes system accounts. Additional users that should not be allowed access to FTP can be added.

In some cases it may be desirable to restrict the access of some users without preventing them completely from using FTP. This can be accomplished be creating /etc/ftpchroot as described in ftpchroot(5). This file lists users and groups subject to FTP access restrictions.

To enable anonymous FTP access to the server, create a user named `ftp` on the FreeBSD system. Users will then be able to log on to the FTP server with a username of `ftp` or `anonymous`. When prompted for the password, any input will be accepted, but by convention, an email address should be used as the password. The FTP server will call chroot(2) when an anonymous user logs in, to restrict access to only the home directory of the `ftp` user.

There are two text files that can be created to specify welcome messages to be displayed to FTP clients. The contents of /etc/ftpwelcome will be displayed to users before they reach the login prompt. After a successful login, the contents of /etc/ftpmotd will be displayed. Note that the path to this file is relative to the login environment, so the contents of ~ftp/etc/ftpmotd would be displayed for anonymous users.

Once the FTP server has been configured, set the appropriate variable in /etc/rc.conf to start the service during boot:

```
ftpd_enable="YES"
```

To start the service now:

```
# service ftpd start
```

Test the connection to the FTP server by typing:

```
% ftp localhost
```

The ftpd daemon uses syslog(3) to log messages. By default, the system log daemon will write messages related to FTP in /var/log/xferlog. The location of the FTP log can be modified by changing the following line in /etc/syslog.conf:

```
ftp.info      /var/log/xferlog
```

Be aware of the potential problems involved with running an anonymous FTP server. In particular, think twice about allowing anonymous users to upload files. It may turn out that the FTP site becomes a forum for the trade of unlicensed commercial software or worse. If anonymous FTP uploads are required, then verify the permissions so that these files cannot be read by other anonymous users until they have been reviewed by an administrator.

33.11. File and Print Services for Microsoft® Windows® Clients (Samba)

Samba is a popular open source software package that provides file and print services using the SMB/CIFS protocol. This protocol is built into Microsoft® Windows® systems. It can be added to non-Microsoft® Windows® systems by installing the Samba client libraries. The protocol allows clients to access shared data and printers. These shares can be mapped as a local disk drive and shared printers can be used as if they were local printers.

On FreeBSD, the Samba client libraries can be installed using the net/samba416 port or package. The client provides the ability for a FreeBSD system to access SMB/CIFS shares in a Microsoft® Windows® network.

A FreeBSD system can also be configured to act as a Samba server by installing the same net/samba416 port or package. This allows the administrator to create SMB/CIFS shares on the FreeBSD system which can be accessed by clients running Microsoft® Windows® or the Samba client libraries.

33.11.1. Server Configuration

Samba is configured in /usr/local/etc/smb4.conf. This file must be created before Samba can be used.

A simple smb4.conf to share directories and printers with Windows® clients in a workgroup is shown here. For more complex setups involving LDAP or Active Directory, it is easier to use samba-tool(8) to create the initial smb4.conf.

```
[global]
workgroup = WORKGROUP
server string = Samba Server Version %v
netbios name = ExampleMachine
wins support = Yes
security = user
passdb backend = tdbsam

# Example: share /usr/src accessible only to 'developer' user
[src]
path = /usr/src
valid users = developer
writable  = yes
browsable = yes
read only = no
guest ok = no
public = no
create mask = 0666
directory mask = 0755
```

Global Settings

Settings that describe the network are added in /usr/local/etc/smb4.conf:

`workgroup`

The name of the workgroup to be served.

`netbios name`

The NetBIOS name by which a Samba server is known. By default, it is the same as the first component of the host's DNS name.

`server string`

The string that will be displayed in the output of `net view` and some other networking tools that seek to display descriptive text about the server.

`wins support`

Whether Samba will act as a WINS server. Do not enable support for WINS on more than one server on the network.

Security Settings

The most important settings in /usr/local/etc/smb4.conf are the security model and the backend password format. These directives control the options:

`security`

If the clients use usernames that are the same as their usernames on the FreeBSD machine, user level security should be used. `security = user` is the default security policy and it requires clients to first log on before they can access shared resources.

Refer to smb.conf(5) to learn about other supported settings for the `security` option.

`passdb backend`

Samba has several different backend authentication models. Clients may be authenticated with LDAP, NIS+, an SQL database, or a modified password file. The recommended authentication method, `tdbsam`, is ideal for simple networks and is covered here. For larger or more complex networks, `ldapsam` is recommended. `smbpasswd` was the former default and is now obsolete.

Samba Users

FreeBSD user accounts must be mapped to the `SambaSAMAccount` database for Windows® clients to access the share. Map existing FreeBSD user accounts using pdbedit(8):

```
# pdbedit -a -u username
```

This section has only mentioned the most commonly used settings. Refer to the Official Samba Wiki for additional information about the available configuration options.

33.11.2. Starting Samba

To enable Samba at boot time, add the following line to /etc/rc.conf:

```
samba_server_enable="YES"
```

To start Samba now:

```
# service samba_server start
Performing sanity check on Samba configuration: OK
Starting nmbd.
Starting smbd.
```

Samba consists of three separate daemons. Both the nmbd and smbd daemons are started by `samba_enable`. If winbind name resolution is also required, set:

```
winbindd_enable="YES"
```

Samba can be stopped at any time by typing:

```
# service samba_server stop
```

Samba is a complex software suite with functionality that allows broad integration with Microsoft® Windows® networks. For more information about functionality beyond the basic configuration described here, refer to https://www.samba.org.

33.12. Clock Synchronization with NTP

Over time, a computer's clock is prone to drift. This is problematic as many network services require the computers on a network to share the same accurate time. Accurate time is also needed to ensure that file timestamps stay consistent. The Network Time Protocol (NTP) is one way to provide clock accuracy in a network.

FreeBSD includes ntpd(8) which can be configured to query other NTP servers to synchronize the clock on that machine or to provide time services to other computers in the network.

This section describes how to configure ntpd on FreeBSD. Further documentation can be found in /usr/share/doc/ntp/ in HTML format.

33.12.1. NTP Configuration

On FreeBSD, the built-in ntpd can be used to synchronize a system's clock. ntpd is configured using rc.conf(5) variables and /etc/ntp.conf, as detailed in the following sections.

ntpd communicates with its network peers using UDP packets. Any firewalls between the machine and its NTP peers must be configured to allow UDP packets in and out on port 123.

The /etc/ntp.conf file

ntpd reads /etc/ntp.conf to determine which NTP servers to query. Choosing several NTP servers is recommended in case one of the servers becomes unreachable or its clock proves unreliable. As ntpd receives responses, it favors reliable servers over the less reliable ones. The servers which are queried can be local to the network, provided by an ISP, or selected from an online list of publicly accessible NTP servers. When choosing a public NTP server, select one that is geographically close and review its usage policy. The `pool` configuration keyword selects one or more servers from a pool of servers. An online list of publicly accessible NTP pools is available, organized by geographic area. In addition, FreeBSD provides a project-sponsored pool, `0.freebsd.pool.ntp.org`.

Example 43. Sample /etc/ntp.conf

This is a simple example of an ntp.conf file. It can safely be used as-is; it contains the recommended `restrict` options for operation on a publicly-accessible network connection.

```
# Disallow ntpq control/query access.  Allow peers to be added only
# based on pool and server statements in this file.
restrict default limited kod nomodify notrap noquery nopeer
restrict source  limited kod nomodify notrap noquery

# Allow unrestricted access from localhost for queries and control.
restrict 127.0.0.1
restrict ::1

# Add a specific server.
server ntplocal.example.com iburst

# Add FreeBSD pool servers until 3-6 good servers are available.
tos minclock 3 maxclock 6
pool 0.freebsd.pool.ntp.org iburst

# Use a local leap-seconds file.
leapfile "/var/db/ntpd.leap-seconds.list"
```

The format of this file is described in ntp.conf(5). The descriptions below provide a quick overview of just the keywords used in the sample file above.

By default, an NTP server is accessible to any network host. The `restrict` keyword controls which systems can access the server. Multiple `restrict` entries are supported, each one refining the restrictions given in previous statements. The values shown in the example grant the local system full query and control access, while allowing remote systems only the ability to query the time. For more details, refer to the `Access Control Support` subsection of ntp.conf(5).

The `server` keyword specifies a single server to query. The file can contain multiple server keywords, with one server listed on each line. The `pool` keyword specifies a pool of servers. ntpd will add one or more servers from this pool as needed to reach the number of peers specified using the `tos minclock` value. The `iburst` keyword directs ntpd to perform a burst of eight quick packet exchanges with a server when contact is first established, to help quickly synchronize system time.

The `leapfile` keyword specifies the location of a file containing information about leap seconds. The file is updated automatically by periodic(8). The file location specified by this keyword must match the location set in the `ntp_db_leapfile` variable in /etc/rc.conf.

NTP entries in /etc/rc.conf

Set `ntpd_enable=YES` to start ntpd at boot time. Once `ntpd_enable=YES` has been added to /etc/rc.conf, ntpd can be started immediately without rebooting the system by typing:

```
# service ntpd start
```

Only `ntpd_enable` must be set to use ntpd. The rc.conf variables listed below may also be set as needed.

Set `ntpd_sync_on_start=YES` to allow ntpd to step the clock any amount, one time at startup. Normally ntpd will log an error message and exit if the clock is off by more than 1000 seconds. This option is especially useful on systems without a battery-backed realtime clock.

Set `ntpd_oomprotect=YES` to protect the ntpd daemon from being killed by the system attempting to recover from an Out Of Memory (OOM) condition.

Set `ntpd_config=` to the location of an alternate ntp.conf file.

Set `ntpd_flags=` to contain any other ntpd flags as needed, but avoid using these flags which are managed internally by /etc/rc.d/ntpd:

- `-p` (pid file location)
- `-c` (set `ntpd_config=` instead)

ntpd and the unprivileged `ntpd` user

ntpd on FreeBSD can start and run as an unprivileged user. Doing so requires the mac_ntpd(4) policy module. The /etc/rc.d/ntpd startup script first examines the NTP configuration. If possible, it loads the `mac_ntpd` module, then starts ntpd as unprivileged user `ntpd` (user id 123). To avoid problems with file and directory access, the startup script will not automatically start ntpd as `ntpd` when the configuration contains any file-related options.

The presence of any of the following in `ntpd_flags` requires manual configuration as described below to run as the `ntpd` user:

- -f or --driftfile
- -i or --jaildir
- -k or --keyfile
- -l or --logfile
- -s or --statsdir

The presence of any of the following keywords in ntp.conf requires manual configuration as described below to run as the `ntpd` user:

- crypto
- driftfile
- key
- logdir
- statsdir

To manually configure ntpd to run as user `ntpd`:

- Ensure that the `ntpd` user has access to all the files and directories specified in the configuration.
- Arrange for the `mac_ntpd` module to be loaded or compiled into the kernel. See mac_ntpd(4) for details.
- Set `ntpd_user="ntpd"` in /etc/rc.conf

33.12.2. Using NTP with a PPP Connection

ntpd does not need a permanent connection to the Internet to function properly. However, if a PPP connection is configured to dial out on demand, NTP traffic should be prevented from triggering a dial out or keeping the connection alive. This can be configured with `filter` directives in /etc/ppp/ppp.conf. For example:

```
set filter dial 0 deny udp src eq 123
# Prevent NTP traffic from initiating dial out
set filter dial 1 permit 0 0
set filter alive 0 deny udp src eq 123
# Prevent incoming NTP traffic from keeping the connection open
set filter alive 1 deny udp dst eq 123
# Prevent outgoing NTP traffic from keeping the connection open
set filter alive 2 permit 0/0 0/0
```

For more details, refer to the `PACKET FILTERING` section in ppp(8) and the examples in /usr/share/examples/ppp/.

> Some Internet access providers block low-numbered ports, preventing NTP from functioning since replies never reach the machine.

33.13. iSCSI Initiator and Target Configuration

iSCSI is a way to share storage over a network. Unlike NFS, which works at the file system level, iSCSI works at the block device level.

In iSCSI terminology, the system that shares the storage is known as the *target*. The storage can be a physical disk, or an area representing multiple disks or a portion of a physical disk. For example, if the disk(s) are formatted with ZFS, a zvol can be created to use as the iSCSI storage.

The clients which access the iSCSI storage are called *initiators*. To initiators, the storage available through iSCSI appears as a raw, unformatted disk known as a LUN. Device nodes for the disk appear in /dev/ and the device must be separately formatted and mounted.

FreeBSD provides a native, kernel-based iSCSI target and initiator. This section describes how to configure a FreeBSD system as a target or an initiator.

33.13.1. Configuring an iSCSI Target

To configure an iSCSI target, create the /etc/ctl.conf configuration file, add a line to /etc/rc.conf to make sure the ctld(8) daemon is automatically started at boot, and then start the daemon.

The following is an example of a simple /etc/ctl.conf configuration file. Refer to ctl.conf(5) for a complete description of this file's available options.

```
portal-group pg0 {
    discovery-auth-group no-authentication
    listen 0.0.0.0
    listen [::]
}

target iqn.2012-06.com.example:target0 {
    auth-group no-authentication
    portal-group pg0

    lun 0 {
        path /data/target0-0
        size 4G
    }
}
```

The first entry defines the `pg0` portal group. Portal groups define which network addresses the ctld(8) daemon will listen on. The `discovery-auth-group no-authentication` entry indicates that any initiator is allowed to perform iSCSI target discovery without authentication. Lines three and four configure ctld(8) to listen on all IPv4 (`listen 0.0.0.0`) and IPv6 (`listen [::]`) addresses on the default port of 3260.

It is not necessary to define a portal group as there is a built-in portal group called `default`. In this case, the difference between `default` and `pg0` is that with `default`, target discovery is always denied, while with `pg0`, it is always allowed.

The second entry defines a single target. Target has two possible meanings: a machine serving iSCSI or a named group of LUNs. This example uses the latter meaning, where `iqn.2012-06.com.example:target0` is the target name. This target name is suitable for testing purposes. For actual use, change `com.example` to the real domain name, reversed. The `2012-06` represents the year and month of acquiring control of that domain name, and `target0` can be any value. Any number of targets can be defined in this configuration file.

The `auth-group no-authentication` line allows all initiators to connect to the specified target and `portal-group pg0` makes the target reachable through the `pg0` portal group.

The next section defines the LUN. To the initiator, each LUN will be visible as a separate disk device. Multiple LUNs can be defined for each target. Each LUN is identified by a number, where LUN 0 is mandatory. The `path /data/target0-0` line defines the full path to a file or zvol backing the LUN. That path must exist before starting ctld(8). The second line is optional and specifies the size of the LUN.

Next, to make sure the ctld(8) daemon is started at boot, add this line to /etc/rc.conf:

```
ctld_enable="YES"
```

To start ctld(8) now, run this command:

```
# service ctld start
```

As the ctld(8) daemon is started, it reads /etc/ctl.conf. If this file is edited after the daemon starts, use this command so that the changes take effect immediately:

```
# service ctld reload
```

Authentication

The previous example is inherently insecure as it uses no authentication, granting anyone full access to all targets. To require a username and password to access targets, modify the configuration as follows:

```
auth-group ag0 {
    chap username1 secretsecret
    chap username2 anothersecret
}

portal-group pg0 {
    discovery-auth-group no-authentication
    listen 0.0.0.0
    listen [::]
}

target iqn.2012-06.com.example:target0 {
    auth-group ag0
    portal-group pg0
    lun 0 {
        path /data/target0-0
        size 4G
    }
}
```

The `auth-group` section defines username and password pairs. An initiator trying to connect to `iqn.2012-06.com.example:target0` must first specify a defined username and secret. However, target discovery is still permitted without authentication. To require target discovery authentication, set `discovery-auth-group` to a defined `auth-group` name instead of `no-authentication`.

It is common to define a single exported target for every initiator. As a shorthand for the syntax above, the username and password can be specified directly in the target entry:

```
target iqn.2012-06.com.example:target0 {
    portal-group pg0
    chap username1 secretsecret

    lun 0 {
        path /data/target0-0
        size 4G
    }
}
```

33.13.2. Configuring an iSCSI Initiator

The iSCSI initiator described in this section is supported starting with FreeBSD 10.0-RELEASE. To use the iSCSI initiator available in older versions, refer to iscontrol(8).

The iSCSI initiator requires that the iscsid(8) daemon is running. This daemon does not use a configuration file. To start it automatically at boot, add this line to /etc/rc.conf:

```
iscsid_enable="YES"
```

To start iscsid(8) now, run this command:

```
# service iscsid start
```

Connecting to a target can be done with or without an /etc/iscsi.conf configuration file. This section demonstrates both types of connections.

Connecting to a Target Without a Configuration File

To connect an initiator to a single target, specify the IP address of the portal and the name of the target:

```
# iscsictl -A -p 10.10.10.10 -t iqn.2012-06.com.example:target0
```

To verify if the connection succeeded, run `iscsictl` without any arguments. The output should look similar to this:

```
Target name                                          Target portal    State
iqn.2012-06.com.example:target0                      10.10.10.10      Connected: da0
```

In this example, the iSCSI session was successfully established, with /dev/da0 representing the attached LUN. If the `iqn.2012-06.com.example:target0` target exports more than one LUN, multiple device nodes will be shown in that section of the output:

```
Connected: da0 da1 da2.
```

Any errors will be reported in the output, as well as the system logs. For example, this message usually means that the iscsid(8) daemon is not running:

```
Target name                                          Target portal    State
```

```
iqn.2012-06.com.example:target0                  10.10.10.10     Waiting for iscsid(8)
```

The following message suggests a networking problem, such as a wrong IP address or port:

```
Target name                                      Target portal   State
iqn.2012-06.com.example:target0                  10.10.10.11     Connection refused
```

This message means that the specified target name is wrong:

```
Target name                                      Target portal   State
iqn.2012-06.com.example:target0                  10.10.10.10     Not found
```

This message means that the target requires authentication:

```
Target name                                      Target portal   State
iqn.2012-06.com.example:target0                  10.10.10.10     Authentication failed
```

To specify a CHAP username and secret, use this syntax:

```
# iscsictl -A -p 10.10.10.10 -t iqn.2012-06.com.example:target0 -u user -s
secretsecret
```

Connecting to a Target with a Configuration File

To connect using a configuration file, create /etc/iscsi.conf with contents like this:

```
t0 {
    TargetAddress   = 10.10.10.10
    TargetName      = iqn.2012-06.com.example:target0
    AuthMethod      = CHAP
    chapIName       = user
    chapSecret      = secretsecret
}
```

The `t0` specifies a nickname for the configuration file section. It will be used by the initiator to specify which configuration to use. The other lines specify the parameters to use during connection. The `TargetAddress` and `TargetName` are mandatory, whereas the other options are optional. In this example, the CHAP username and secret are shown.

To connect to the defined target, specify the nickname:

```
# iscsictl -An t0
```

Alternately, to connect to all targets defined in the configuration file, use:

```
# iscsictl -Aa
```

To make the initiator automatically connect to all targets in /etc/iscsi.conf, add the following to /etc/rc.conf:

```
iscsictl_enable="YES"
iscsictl_flags="-Aa"
```

Chapter 34. Firewalls

34.1. Synopsis

Firewalls make it possible to filter the incoming and outgoing traffic that flows through a system. A firewall can use one or more sets of "rules" to inspect network packets as they come in or go out of network connections and either allows the traffic through or blocks it. The rules of a firewall can inspect one or more characteristics of the packets such as the protocol type, source or destination host address, and source or destination port.

Firewalls can enhance the security of a host or a network. They can be used to do one or more of the following:

- Protect and insulate the applications, services, and machines of an internal network from unwanted traffic from the public Internet.
- Limit or disable access from hosts of the internal network to services of the public Internet.
- Support network address translation (NAT), which allows an internal network to use private IP addresses and share a single connection to the public Internet using either a single IP address or a shared pool of automatically assigned public addresses.

FreeBSD has three firewalls built into the base system: PF, IPFW, and IPFILTER, also known as IPF. FreeBSD also provides two traffic shapers for controlling bandwidth usage: altq(4) and dummynet(4). ALTQ has traditionally been closely tied with PF and dummynet with IPFW. Each firewall uses rules to control the access of packets to and from a FreeBSD system, although they go about it in different ways and each has a different rule syntax.

FreeBSD provides multiple firewalls in order to meet the different requirements and preferences for a wide variety of users. Each user should evaluate which firewall best meets their needs.

Read this chapter to learn:

- How to define packet filtering rules.
- The differences between the firewalls built into FreeBSD.
- How to use and configure the PF firewall.
- How to use and configure the IPFW firewall.
- How to use and configure the IPFILTER firewall.

Before reading this chapter:

- Understand basic FreeBSD and Internet concepts.

Since all firewalls are based on inspecting the values of selected packet control fields, the creator of the firewall ruleset must have an understanding of how TCP/IP works, what the different values in the packet control fields are, and how these values are used in a normal session conversation. For a good introduction, refer to Daryl's TCP/IP Primer.

34.2. Firewall Concepts

A ruleset contains a group of rules which pass or block packets based on the values contained in the packet. The bi-directional exchange of packets between hosts comprises a session conversation. The firewall ruleset processes both the packets arriving from the public Internet, as well as the packets produced by the system as a response to them. Each TCP/IP service is predefined by its protocol and listening port. Packets destined for a specific service originate from the source address using an unprivileged port and target the specific service port on the destination address. All the above parameters can be used as selection criteria to create rules which will pass or block services.

To lookup unknown port numbers, refer to /etc/services. Alternatively, visit https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers and do a port number lookup to find the purpose of a particular port number.

Check out this link for port numbers used by Trojans.

FTP has two modes: active mode and passive mode. The difference is in how the data channel is acquired. Passive mode is more secure as the data channel is acquired by the ordinal ftp session requester. For a good explanation of FTP and the different modes, see http://www.slacksite.com/other/ftp.html.

A firewall ruleset can be either "exclusive" or "inclusive". An exclusive firewall allows all traffic through except for the traffic matching the ruleset. An inclusive firewall does the reverse as it only allows traffic matching the rules through and blocks everything else.

An inclusive firewall offers better control of the outgoing traffic, making it a better choice for systems that offer services to the public Internet. It also controls the type of traffic originating from the public Internet that can gain access to a private network. All traffic that does not match the rules is blocked and logged. Inclusive firewalls are generally safer than exclusive firewalls because they significantly reduce the risk of allowing unwanted traffic.

> Unless noted otherwise, all configuration and example rulesets in this chapter create inclusive firewall rulesets.

Security can be tightened further using a "stateful firewall". This type of firewall keeps track of open connections and only allows traffic which either matches an existing connection or opens a new, allowed connection.

Stateful filtering treats traffic as a bi-directional exchange of packets comprising a session. When state is specified on a matching rule the firewall dynamically generates internal rules for each anticipated packet being exchanged during the session. It has sufficient matching capabilities to determine if a packet is valid for a session. Any packets that do not properly fit the session template are automatically rejected.

When the session completes, it is removed from the dynamic state table.

Stateful filtering allows one to focus on blocking/passing new sessions. If the new session is passed, all its subsequent packets are allowed automatically and any impostor packets are automatically rejected. If a new session is blocked, none of its subsequent packets are allowed. Stateful filtering provides advanced matching abilities capable of defending against the flood of different attack methods employed by attackers.

NAT stands for *Network Address Translation*. NAT function enables the private LAN behind the firewall to share a single ISP-assigned IP address, even if that address is dynamically assigned. NAT allows each computer in the LAN to have Internet access, without having to pay the ISP for multiple Internet accounts or IP addresses.

NAT will automatically translate the private LAN IP address for each system on the LAN to the single public IP address as packets exit the firewall bound for the public Internet. It also performs the reverse translation for returning packets.

According to RFC 1918, the following IP address ranges are reserved for private networks which will never be routed directly to the public Internet, and therefore are available for use with NAT:

- `10.0.0.0/8`.
- `172.16.0.0/12`.
- `192.168.0.0/16`.

When working with the firewall rules, be *very careful*. Some configurations *can lock the administrator out* of the server. To be on the safe side, consider performing the initial firewall configuration from the local console rather than doing it remotely over ssh.

34.3. PF

Since FreeBSD 5.3, a ported version of OpenBSD's PF firewall has been included as an integrated part of the base system. PF is a complete, full-featured firewall that has optional support for ALTQ (Alternate Queuing), which provides Quality of Service (QoS).

The OpenBSD Project maintains the definitive reference for PF in the PF FAQ. Peter Hansteen maintains a thorough PF tutorial at http://home.nuug.no/~peter/pf/.

When reading the PF FAQ, keep in mind that FreeBSD's version of PF has diverged substantially from the upstream OpenBSD version over the years. Not all features work the same way on FreeBSD as they do in OpenBSD and vice versa.

The FreeBSD packet filter mailing list is a good place to ask questions about configuring and running the PF firewall. Check the mailing list archives before asking a question as it may have already been answered.

This section of the Handbook focuses on PF as it pertains to FreeBSD. It demonstrates how to enable PF and ALTQ. It also provides several examples for creating rulesets on a FreeBSD system.

34.3.1. Enabling PF

To use PF, its kernel module must be first loaded. This section describes the entries that can be added to /etc/rc.conf to enable PF.

Start by adding `pf_enable=yes` to /etc/rc.conf:

```
# sysrc pf_enable=yes
```

Additional options, described in pfctl(8), can be passed to PF when it is started. Add or change this entry in /etc/rc.conf and specify any required flags between the two quotes (`""`):

```
pf_flags=""                     # additional flags for pfctl startup
```

PF will not start if it cannot find its ruleset configuration file. By default, FreeBSD does not ship with a ruleset and there is no /etc/pf.conf. Example rulesets can be found in /usr/share/examples/pf/. If a custom ruleset has been saved somewhere else, add a line to /etc/rc.conf which specifies the full path to the file:

```
pf_rules="/path/to/pf.conf"
```

Logging support for PF is provided by pflog(4). To enable logging support, add `pflog_enable=yes` to /etc/rc.conf:

```
# sysrc pflog_enable=yes
```

The following lines can also be added to change the default location of the log file or to specify any additional flags to pass to pflog(4) when it is started:

```
pflog_logfile="/var/log/pflog"  # where pflogd should store the logfile
pflog_flags=""                  # additional flags for pflogd startup
```

Finally, if there is a LAN behind the firewall and packets need to be forwarded for the computers on the LAN, or NAT is required, enable the following option:

```
gateway_enable="YES"            # Enable as LAN gateway
```

After saving the needed edits, PF can be started with logging support by typing:

```
# service pf start
# service pflog start
```

By default, PF reads its configuration rules from /etc/pf.conf and modifies, drops, or passes packets according to the rules or definitions specified in this file. The FreeBSD installation includes several sample files located in /usr/share/examples/pf/. Refer to the PF FAQ for complete coverage of PF rulesets.

To control PF, use `pfctl`. Useful `pfctl` Options summarizes some useful options to this command. Refer to pfctl(8) for a description of all available options:

Table 49. Useful `pfctl` Options

Command	**Purpose**
`pfctl -e`	Enable PF.
`pfctl -d`	Disable PF.
`pfctl -F all -f /etc/pf.conf`	Flush all NAT, filter, state, and table rules and reload /etc/pf.conf.
`pfctl -s [ rules | nat | states ]`	Report on the filter rules, NAT rules, or state table.
`pfctl -vnf /etc/pf.conf`	Check /etc/pf.conf for errors, but do not load ruleset.

security/sudo is useful for running commands like `pfctl` that require elevated privileges. It can be installed from the Ports Collection.

To keep an eye on the traffic that passes through the PF firewall, consider installing the sysutils/pftop package or port. Once installed, pftop can be run to view a running snapshot of traffic in a format which is similar to top(1).

34.3.2. PF Rulesets

This section demonstrates how to create a customized ruleset. It starts with the simplest of rulesets and builds upon its concepts using several examples to demonstrate real-world usage of PF's many features.

The simplest possible ruleset is for a single machine that does not run any services and which needs access to one network, which may be the Internet. To create this minimal ruleset, edit /etc/pf.conf so it looks like this:

```
block in all
pass out all keep state
```

The first rule denies all incoming traffic by default. The second rule allows connections created by this system to pass out, while retaining state information on those connections. This state information allows return traffic for those connections to pass back and should only be used on machines that can be trusted. The ruleset can be loaded with:

```
# pfctl -e ; pfctl -f /etc/pf.conf
```

In addition to keeping state, PF provides *lists* and *macros* which can be defined for use when creating rules. Macros can include lists and need to be defined before use. As an example, insert these lines at the very top of the ruleset:

```
tcp_services = "{ ssh, smtp, domain, www, pop3, auth, pop3s }"
udp_services = "{ domain }"
```

PF understands port names as well as port numbers, as long as the names are listed in /etc/services. This example creates two macros. The first is a list of seven TCP port names and the second is one UDP port name. Once defined, macros can be used in rules. In this example, all traffic is blocked except for the connections initiated by this system for the seven specified TCP services and the one specified UDP service:

```
tcp_services = "{ ssh, smtp, domain, www, pop3, auth, pop3s }"
udp_services = "{ domain }"
block all
pass out proto tcp to any port $tcp_services keep state
pass proto udp to any port $udp_services keep state
```

Even though UDP is considered to be a stateless protocol, PF is able to track some state information. For example, when a UDP request is passed which asks a name server about a domain name, PF will watch for the response to pass it back.

Whenever an edit is made to a ruleset, the new rules must be loaded so they can be used:

```
# pfctl -f /etc/pf.conf
```

If there are no syntax errors, `pfctl` will not output any messages during the rule load. Rules can also be tested before attempting to load them:

```
# pfctl -nf /etc/pf.conf
```

Including `-n` causes the rules to be interpreted only, but not loaded. This provides an opportunity to correct any errors. At all times, the last valid ruleset loaded will be enforced until either PF is disabled or a new ruleset is loaded.

> Adding `-v` to a `pfctl` ruleset verify or load will display the fully parsed rules exactly the way they will be loaded. This is extremely useful when debugging rules.

A Simple Gateway with NAT

This section demonstrates how to configure a FreeBSD system running PF to act as a gateway for at least one other machine. The gateway needs at least two network interfaces, each connected to a separate network. In this example, xl0 is connected to the Internet and xl1 is connected to the internal network.

First, enable the gateway to let the machine forward the network traffic it receives on one interface to another interface. This sysctl setting will forward IPv4 packets:

```
# sysctl net.inet.ip.forwarding=1
```

To forward IPv6 traffic, use:

```
# sysctl net.inet6.ip6.forwarding=1
```

To enable these settings at system boot, use sysrc(8) to add them to /etc/rc.conf:

```
# sysrc gateway_enable=yes
# sysrc ipv6_gateway_enable=yes
```

Verify with `ifconfig` that both of the interfaces are up and running.

Next, create the PF rules to allow the gateway to pass traffic. While the following rule allows stateful traffic from hosts of the internal network to pass to the gateway, the `to` keyword does not guarantee passage all the way from source to destination:

```
pass in on xl1 from xl1:network to xl0:network port $ports keep state
```

That rule only lets the traffic pass in to the gateway on the internal interface. To let the packets go further, a matching rule is needed:

```
pass out on xl0 from xl1:network to xl0:network port $ports keep state
```

While these two rules will work, rules this specific are rarely needed. For a busy network admin, a readable ruleset is a safer ruleset. The remainder of this section demonstrates how to keep the rules as simple as possible for readability. For example, those two rules could be replaced with one rule:

```
pass from xl1:network to any port $ports keep state
```

The `interface:network` notation can be replaced with a macro to make the ruleset even more readable. For example, a `$localnet` macro could be defined as the network directly attached to the internal interface (`$xl1:network`). Alternatively, the definition of `$localnet` could be changed to an *IP address/netmask* notation to denote a network, such as `192.168.100.1/24` for a subnet of private addresses.

If required, `$localnet` could even be defined as a list of networks. Whatever the specific needs, a sensible `$localnet` definition could be used in a typical pass rule as follows:

```
pass from $localnet to any port $ports keep state
```

The following sample ruleset allows all traffic initiated by machines on the internal network. It first defines two macros to represent the external and internal 3COM interfaces of the gateway.

For dialup users, the external interface will use tun0. For an ADSL connection, specifically those using PPP over Ethernet (PPPoE), the correct external interface is tun0, not the physical Ethernet interface.

```
ext_if = "xl0"  # macro for external interface - use tun0 for PPPoE
int_if = "xl1"  # macro for internal interface
localnet = $int_if:network
# ext_if IP address could be dynamic, hence ($ext_if)
nat on $ext_if from $localnet to any -> ($ext_if)
block all
pass from { lo0, $localnet } to any keep state
```

This ruleset introduces the `nat` rule which is used to handle the network address translation from the non-routable addresses inside the internal network to the IP address assigned to the external interface. The parentheses surrounding the last part of the nat rule `($ext_if)` is included when the IP address of the external interface is dynamically assigned. It ensures that network traffic runs without serious interruptions even if the external IP address changes.

Note that this ruleset probably allows more traffic to pass out of the network than is needed. One reasonable setup could create this macro:

```
client_out = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
    https, cvspserver, 2628, 5999, 8000, 8080 }"
```

to use in the main pass rule:

```
pass inet proto tcp from $localnet to any port $client_out \
    flags S/SA keep state
```

A few other pass rules may be needed. This one enables SSH on the external interface:

```
pass in inet proto tcp to $ext_if port ssh
```

This macro definition and rule allows DNS and NTP for internal clients:

```
udp_services = "{ domain, ntp }"
pass quick inet proto { tcp, udp } to any port $udp_services keep state
```

Note the `quick` keyword in this rule. Since the ruleset consists of several rules, it is important to understand the relationships between the rules in a ruleset. Rules are evaluated from top to bottom, in the sequence they are written. For each packet or connection evaluated by PF, *the last matching rule* in the ruleset is the one which is applied. However, when a packet matches a rule which contains the `quick` keyword, the rule processing stops and the packet is treated according to that rule. This is very useful when an exception to the general rules is needed.

Creating an FTP Proxy

Configuring working FTP rules can be problematic due to the nature of the FTP protocol. FTP predates firewalls by several decades and is insecure in its design. The most common points against using FTP include:

- Passwords are transferred in the clear.
- The protocol demands the use of at least two TCP connections (control and data) on separate ports.
- When a session is established, data is communicated using randomly selected ports.

All of these points present security challenges, even before considering any potential security weaknesses in client or server software. More secure alternatives for file transfer exist, such as sftp(1) or scp(1), which both feature authentication and data transfer over encrypted connections.

For those situations when FTP is required, PF provides redirection of FTP traffic to a small proxy program called ftp-proxy(8), which is included in the base system of FreeBSD. The role of the proxy is to dynamically insert and delete rules in the ruleset, using a set of anchors, to correctly handle FTP traffic.

To enable the FTP proxy, add this line to /etc/rc.conf:

```
ftpproxy_enable="YES"
```

Then start the proxy by running:

```
# service ftp-proxy start
```

For a basic configuration, three elements need to be added to /etc/pf.conf. First, the anchors which the proxy will use to insert the rules it generates for the FTP sessions:

```
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
```

Second, a pass rule is needed to allow FTP traffic in to the proxy.

Third, redirection and NAT rules need to be defined before the filtering rules. Insert this `rdr` rule immediately after the `nat` rule:

```
rdr pass on $int_if proto tcp from any to any port ftp -> 127.0.0.1 port 8021
```

Finally, allow the redirected traffic to pass:

```
pass out proto tcp from $proxy to any port ftp
```

where `$proxy` expands to the address the proxy daemon is bound to.

Save /etc/pf.conf, load the new rules, and verify from a client that FTP connections are working:

```
# pfctl -f /etc/pf.conf
```

This example covers a basic setup where the clients in the local network need to contact FTP servers elsewhere. This basic configuration should work well with most combinations of FTP clients and servers. As shown in ftp-proxy(8), the proxy's behavior can be changed in various ways by adding options to the `ftpproxy_flags=` line. Some clients or servers may have specific quirks that must be compensated for in the configuration, or there may be a need to integrate the proxy in specific ways such as assigning FTP traffic to a specific queue.

For ways to run an FTP server protected by PF and ftp-proxy(8), configure a separate `ftp-proxy` in reverse mode, using `-R`, on a separate port with its own redirecting pass rule.

Managing ICMP

Many of the tools used for debugging or troubleshooting a TCP/IP network rely on the Internet Control Message Protocol (ICMP), which was designed specifically with debugging in mind.

The ICMP protocol sends and receives *control messages* between hosts and gateways, mainly to provide feedback to a sender about any unusual or difficult conditions enroute to the target host. Routers use ICMP to negotiate packet sizes and other transmission parameters in a process often referred to as *path MTU discovery*.

From a firewall perspective, some ICMP control messages are vulnerable to known attack vectors. Also, letting all diagnostic traffic pass unconditionally makes debugging easier, but it also makes it easier for others to extract information about the network. For these reasons, the following rule may not be optimal:

```
pass inet proto icmp from any to any
```

One solution is to let all ICMP traffic from the local network through while stopping all probes from outside the network:

```
pass inet proto icmp from $localnet to any keep state
pass inet proto icmp from any to $ext_if keep state
```

Additional options are available which demonstrate some of PF's flexibility. For example, rather than allowing all ICMP messages, one can specify the messages used by ping(8) and traceroute(8). Start by defining a macro for that type of message:

```
icmp_types = "echoreq"
```

and a rule which uses the macro:

```
pass inet proto icmp all icmp-type $icmp_types keep state
```

If other types of ICMP packets are needed, expand `icmp_types` to a list of those packet types. Type `more /usr/src/sbin/pfctl/pfctl_parser.c` to see the list of ICMP message types supported by PF. Refer to http://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml for an explanation of each message type.

Since Unix `traceroute` uses UDP by default, another rule is needed to allow Unix `traceroute`:

```
# allow out the default range for traceroute(8):
pass out on $ext_if inet proto udp from any to any port 33433 >< 33626 keep state
```

Since `TRACERT.EXE` on Microsoft Windows systems uses ICMP echo request messages, only the first rule is needed to allow network traces from those systems. Unix `traceroute` can be instructed to use other protocols as well, and will use ICMP echo request messages if `-I` is used. Check the traceroute(8) man page for details.

Path MTU Discovery

Internet protocols are designed to be device independent, and one consequence of device independence is that the optimal packet size for a given connection cannot always be predicted reliably. The main constraint on packet size is the *Maximum Transmission Unit* (MTU) which sets the upper limit on the packet size for an interface. Type `ifconfig` to view the MTUs for a system's network interfaces.

TCP/IP uses a process known as path MTU discovery to determine the right packet size for a connection. This process sends packets of varying sizes with the "Do not fragment" flag set, expecting an ICMP return packet of "type 3, code 4" when the upper limit has been reached. Type 3 means "destination unreachable", and code 4 is short for "fragmentation needed, but the do-not-fragment flag is set". To allow path MTU discovery in order to support connections to other MTUs, add the `destination unreachable` type to the `icmp_types` macro:

```
icmp_types = "{ echoreq, unreach }"
```

Since the pass rule already uses that macro, it does not need to be modified to support the new ICMP type:

```
pass inet proto icmp all icmp-type $icmp_types keep state
```

PF allows filtering on all variations of ICMP types and codes. The list of possible types and codes are documented in icmp(4) and icmp6(4).

Using Tables

Some types of data are relevant to filtering and redirection at a given time, but their definition is too long to be included in the ruleset file. PF supports the use of tables, which are defined lists that can be manipulated without needing to reload the entire ruleset, and which can provide fast lookups. Table names are always enclosed within `< >`, like this:

```
table <clients> { 192.168.2.0/24, !192.168.2.5 }
```

In this example, the `192.168.2.0/24` network is part of the table, except for the address `192.168.2.5`, which is excluded using the `!` operator. It is also possible to load tables from files where each item is on a separate line, as seen in this example /etc/clients:

```
192.168.2.0/24
!192.168.2.5
```

To refer to the file, define the table like this:

```
table <clients> persist file "/etc/clients"
```

Once the table is defined, it can be referenced by a rule:

```
pass inet proto tcp from <clients> to any port $client_out flags S/SA keep state
```

A table's contents can be manipulated live, using `pfctl`. This example adds another network to the table:

```
# pfctl -t clients -T add 192.168.1.0/16
```

Note that any changes made this way will take affect now, making them ideal for testing, but will not survive a power failure or reboot. To make the changes permanent, modify the definition of the table in the ruleset or edit the file that the table refers to. One can maintain the on-disk copy of the table using a cron(8) job which dumps the table's contents to disk at regular intervals, using a command such as `pfctl -t clients -T show >/etc/clients`. Alternatively, /etc/clients can be updated with the in-memory table contents:

```
# pfctl -t clients -T replace -f /etc/clients
```

Using Overload Tables to Protect SSH

Those who run SSH on an external interface have probably seen something like this in the authentication logs:

```
Sep 26 03:12:34 skapet sshd[25771]: Failed password for root from 200.72.41.31 port
40992 ssh2
Sep 26 03:12:34 skapet sshd[5279]: Failed password for root from 200.72.41.31 port
40992 ssh2
Sep 26 03:12:35 skapet sshd[5279]: Received disconnect from 200.72.41.31: 11: Bye Bye
Sep 26 03:12:44 skapet sshd[29635]: Invalid user admin from 200.72.41.31
Sep 26 03:12:44 skapet sshd[24703]: input_userauth_request: invalid user admin
Sep 26 03:12:44 skapet sshd[24703]: Failed password for invalid user admin from
200.72.41.31 port 41484 ssh2
```

This is indicative of a brute force attack where somebody or some program is trying to discover the user name and password which will let them into the system.

If external SSH access is needed for legitimate users, changing the default port used by SSH can offer some protection. However, PF provides a more elegant solution. Pass rules can contain limits on what connecting hosts can do and violators can be banished to a table of addresses which are denied some or all access. It is even possible to drop all existing connections from machines which overreach the limits.

To configure this, create this table in the tables section of the ruleset:

```
table <bruteforce> persist
```

Then, somewhere early in the ruleset, add rules to block brute access while allowing legitimate access:

```
block quick from <bruteforce>
pass inet proto tcp from any to $localnet port $tcp_services \
    flags S/SA keep state \
    (max-src-conn 100, max-src-conn-rate 15/5, \
    overload <bruteforce> flush global)
```

The part in parentheses defines the limits and the numbers should be changed to meet local requirements. It can be read as follows:

`max-src-conn` is the number of simultaneous connections allowed from one host.

`max-src-conn-rate` is the rate of new connections allowed from any single host (*15*) per number of seconds (*5*).

`overload <bruteforce>` means that any host which exceeds these limits gets its address added to the `bruteforce` table. The ruleset blocks all traffic from addresses in the `bruteforce` table.

Finally, `flush global` says that when a host reaches the limit, that all (`global`) of that host's connections will be terminated (`flush`).

These rules will *not* block slow bruteforcers, as described in http://home.nuug.no/~peter/hailmary2013/.

This example ruleset is intended mainly as an illustration. For example, if a generous number of connections in general are wanted, but the desire is to be more restrictive when it comes to ssh, supplement the rule above with something like the one below, early on in the rule set:

```
pass quick proto { tcp, udp } from any to any port ssh \
    flags S/SA keep state \
    (max-src-conn 15, max-src-conn-rate 5/3, \
    overload <bruteforce> flush global)
```

It May Not be Necessary to Block All Overloaders:

It is worth noting that the overload mechanism is a general technique which does not apply exclusively to SSH, and it is not always optimal to entirely block all traffic from offenders.

For example, an overload rule could be used to protect a mail service or a web service, and the overload table could be used in a rule to assign offenders to a queue with a minimal bandwidth allocation or to redirect to a specific web page.

Over time, tables will be filled by overload rules and their size will grow incrementally, taking up more memory. Sometimes an IP address that is blocked is a dynamically assigned one, which has since been assigned to a host who has a legitimate reason to communicate with hosts in the local network.

For situations like these, pfctl provides the ability to expire table entries. For example, this command will remove `<bruteforce>` table entries which have not been referenced for `86400` seconds:

```
# pfctl -t bruteforce -T expire 86400
```

Similar functionality is provided by security/expiretable, which removes table entries which have not been accessed for a specified period of time.

Once installed, expiretable can be run to remove `<bruteforce>` table entries older than a specified age. This example removes all entries older than 24 hours:

```
/usr/local/sbin/expiretable -v -d -t 24h bruteforce
```

Protecting Against SPAM

Not to be confused with the spamd daemon which comes bundled with spamassassin, mail/spamd can be configured with PF to provide an outer defense against SPAM. This spamd hooks into the PF configuration using a set of redirections.

Spammers tend to send a large number of messages, and SPAM is mainly sent from a few spammer friendly networks and a large number of hijacked machines, both of which are reported to *blocklists* fairly quickly.

When an SMTP connection from an address in a blocklist is received, spamd presents its banner and immediately switches to a mode where it answers SMTP traffic one byte at a time. This technique, which is intended to waste as much time as possible on the spammer's end, is called *tarpitting*. The specific implementation which uses one byte SMTP replies is often referred to as *stuttering*.

This example demonstrates the basic procedure for setting up spamd with automatically updated blocklists. Refer to the man pages which are installed with mail/spamd for more information.

Procedure: Configuring spamd

1. Install the mail/spamd package or port. To use spamd's greylisting features, fdescfs(5) must be mounted at /dev/fd. Add the following line to /etc/fstab:

   ```
   fdescfs /dev/fd fdescfs rw 0 0
   ```

   Then, mount the filesystem:

   ```
   #  mount fdescfs
   ```

2. Next, edit the PF ruleset to include:

   ```
   table <spamd> persist
   table <spamd-white> persist
   rdr pass on $ext_if inet proto tcp from <spamd> to \
       { $ext_if, $localnet } port smtp -> 127.0.0.1 port 8025
   rdr pass on $ext_if inet proto tcp from !<spamd-white> to \
       { $ext_if, $localnet } port smtp -> 127.0.0.1 port 8025
   ```

   The two tables `<spamd>` and `<spamd-white>` are essential. SMTP traffic from an address listed in `<spamd>` but not in `<spamd-white>` is redirected to the spamd daemon listening at port 8025.

3. The next step is to configure spamd in /usr/local/etc/spamd.conf and to add some rc.conf parameters.

   The installation of mail/spamd includes a sample configuration file (/usr/local/etc/spamd.conf.sample) and a man page for spamd.conf. Refer to these for additional configuration options beyond those shown in this example.

   One of the first lines in the configuration file that does not begin with a `#` comment sign contains the block which defines the `all` list, which specifies the lists to use:

```
all:\
    :traplist:allowlist:
```

This entry adds the desired blocklists, separated by colons (`:`). To use an allowlist to subtract addresses from a blocklist, add the name of the allowlist *immediately* after the name of that blocklist. For example: `:blocklist:allowlist:`.

This is followed by the specified blocklist's definition:

```
traplist:\
    :black:\
    :msg="SPAM. Your address %A has sent spam within the last 24 hours":\
    :method=http:\
    :file=www.openbsd.org/spamd/traplist.gz
```

where the first line is the name of the blocklist and the second line specifies the list type. The `msg` field contains the message to display to blocklisted senders during the SMTP dialogue. The `method` field specifies how spamd-setup fetches the list data; supported methods are `http`, `ftp`, from a `file` in a mounted file system, and via `exec` of an external program. Finally, the `file` field specifies the name of the file spamd expects to receive.

The definition of the specified allowlist is similar, but omits the `msg` field since a message is not needed:

```
allowlist:\
    :white:\
    :method=file:\
    :file=/var/mail/allowlist.txt
```

Choose Data Sources with Care:

Using all the blocklists in the sample spamd.conf will block large blocks of the Internet. Administrators need to edit the file to create an optimal configuration which uses applicable data sources and, when necessary, uses custom lists.

Next, add this entry to /etc/rc.conf. Additional flags are described in the man page specified by the comment:

```
spamd_flags="-v" # use "" and see spamd-setup(8) for flags
```

When finished, reload the ruleset, start spamd by typing `service obspamd start`, and complete the configuration using `spamd-setup`. Finally, create a cron(8) job which calls `spamd-setup` to update the tables at reasonable intervals.

On a typical gateway in front of a mail server, hosts will soon start getting trapped within a few seconds to several minutes.

PF also supports *greylisting*, which temporarily rejects messages from unknown hosts with *45n* codes. Messages from greylisted hosts which try again within a reasonable time are let through. Traffic from senders which are set up to behave within the limits set by RFC 1123 and RFC 2821 are immediately let through.

More information about greylisting as a technique can be found at the greylisting.org web site. The most amazing thing about greylisting, apart from its simplicity, is that it still works. Spammers and malware writers have been very slow to adapt to bypass this technique.

The basic procedure for configuring greylisting is as follows:

Procedure: Configuring Greylisting

1. Make sure that fdescfs(5) is mounted as described in Step 1 of the previous Procedure.
2. To run spamd in greylisting mode, add this line to /etc/rc.conf:

   ```
   spamd_grey="YES"  # use spamd greylisting if YES
   ```

   Refer to the spamd man page for descriptions of additional related parameters.
3. To complete the greylisting setup:

   ```
   #  service obspamd restart
   #  service obspamlogd start
   ```

Behind the scenes, the spamdb database tool and the spamlogd whitelist updater perform essential functions for the greylisting feature. spamdb is the administrator’s main interface to managing the block, grey, and allow lists via the contents of the /var/db/spamdb database.

Network Hygiene

This section describes how `block-policy`, `scrub`, and `antispoof` can be used to make the ruleset behave sanely.

The `block-policy` is an option which can be set in the `options` part of the ruleset, which precedes the redirection and filtering rules. This option determines which feedback, if any, PF sends to hosts that are blocked by a rule. The option has two possible values: `drop` drops blocked packets with no feedback, and `return` returns a status code such as `Connection refused`.

If not set, the default policy is `drop`. To change the `block-policy`, specify the desired value:

```
set block-policy return
```

In PF, `scrub` is a keyword which enables network packet normalization. This process reassembles fragmented packets and drops TCP packets that have invalid flag combinations. Enabling `scrub` provides a measure of protection against certain kinds of attacks based on incorrect handling of packet fragments. A number of options are available, but the simplest form is suitable for most configurations:

```
scrub in all
```

Some services, such as NFS, require specific fragment handling options. Refer to https://home.nuug.no/~peter/pf/en/scrub.html for more information.

This example reassembles fragments, clears the "do not fragment" bit, and sets the maximum segment size to 1440 bytes:

```
scrub in all fragment reassemble no-df max-mss 1440
```

The `antispoof` mechanism protects against activity from spoofed or forged IP addresses, mainly by blocking packets appearing on interfaces and in directions which are logically not possible.

These rules weed out spoofed traffic coming in from the rest of the world as well as any spoofed packets which originate in the local network:

```
antispoof for $ext_if
antispoof for $int_if
```

Handling Non-Routable Addresses

Even with a properly configured gateway to handle network address translation, one may have to compensate for other people’s misconfigurations. A common misconfiguration is to let traffic with non-routable addresses out to the Internet. Since traffic from non-routeable addresses can play a part in several DoS attack techniques, consider explicitly blocking traffic from non-routeable addresses from entering the network through the external interface.

In this example, a macro containing non-routable addresses is defined, then used in blocking rules. Traffic to and from these addresses is quietly dropped on the gateway’s external interface.

```
martians = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, \
          10.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, \
          0.0.0.0/8, 240.0.0.0/4 }"

block drop in quick on $ext_if from $martians to any
block drop out quick on $ext_if from any to $martians
```

34.3.3. Enabling ALTQ

On FreeBSD, ALTQ can be used with PF to provide Quality of Service (QOS). Once ALTQ is enabled, queues can be defined in the ruleset which determine the processing priority of outbound packets.

Before enabling ALTQ, refer to altq(4) to determine if the drivers for the network cards installed on the system support it.

ALTQ is not available as a loadable kernel module. If the system's interfaces support ALTQ, create a custom kernel using the instructions in Configuring the FreeBSD Kernel. The following kernel options are available. The first is needed to enable ALTQ. At least one of the other options is necessary to specify the queueing scheduler algorithm:

```
options         ALTQ
options         ALTQ_CBQ        # Class Based Queuing (CBQ)
options         ALTQ_RED        # Random Early Detection (RED)
options         ALTQ_RIO        # RED In/Out
options         ALTQ_HFSC       # Hierarchical Packet Scheduler (HFSC)
options         ALTQ_PRIQ       # Priority Queuing (PRIQ)
```

The following scheduler algorithms are available:

CBQ

Class Based Queuing (CBQ) is used to divide a connection's bandwidth into different classes or queues to prioritize traffic based on filter rules.

RED

Random Early Detection (RED) is used to avoid network congestion by measuring the length of the queue and comparing it to the minimum and maximum thresholds for the queue. When the queue is over the maximum, all new packets are randomly dropped.

RIO

In Random Early Detection In and Out (RIO) mode, RED maintains multiple average queue lengths and multiple threshold values, one for each QOS level.

HFSC

Hierarchical Fair Service Curve Packet Scheduler (HFSC) is described in http://www-2.cs.cmu.edu/~hzhang/HFSC/main.html.

PRIQ

Priority Queuing (PRIQ) always passes traffic that is in a higher queue first.

More information about the scheduling algorithms and example rulesets are available at the OpenBSD's web archive.

34.4. IPFW

IPFW is a stateful firewall written for FreeBSD which supports both IPv4 and IPv6. It is comprised of several components: the kernel firewall filter rule processor and its integrated packet accounting facility, the logging facility, NAT, the dummynet(4) traffic shaper, a forward facility, a bridge facility, and an ipstealth facility.

FreeBSD provides a sample ruleset in /etc/rc.firewall which defines several firewall types for common scenarios to assist novice users in generating an appropriate ruleset. IPFW provides a powerful syntax which advanced users can use to craft customized rulesets that meet the security requirements of a given environment.

This section describes how to enable IPFW, provides an overview of its rule syntax, and demonstrates several rulesets for common configuration scenarios.

34.4.1. Enabling IPFW

IPFW is included in the basic FreeBSD install as a kernel loadable module, meaning that a custom kernel is not needed in order to enable IPFW.

For those users who wish to statically compile IPFW support into a custom kernel, see IPFW Kernel Options.

To configure the system to enable IPFW at boot time, add `firewall_enable="YES"` to /etc/rc.conf:

```
# sysrc firewall_enable="YES"
```

To use one of the default firewall types provided by FreeBSD, add another line which specifies the type:

```
# sysrc firewall_type="open"
```

The available types are:

- `open`: passes all traffic.
- `client`: protects only this machine.
- `simple`: protects the whole network.
- `closed`: entirely disables IP traffic except for the loopback interface.
- `workstation`: protects only this machine using stateful rules.
- `UNKNOWN`: disables the loading of firewall rules.
- filename: full path of the file containing the firewall ruleset.

If `firewall_type` is set to either `client` or `simple`, modify the default rules found in /etc/rc.firewall to fit the configuration of the system.

Note that the `filename` type is used to load a custom ruleset.

An alternate way to load a custom ruleset is to set the `firewall_script` variable to the absolute path of an *executable script* that includes IPFW commands. The examples used in this section assume that the `firewall_script` is set to /etc/ipfw.rules:

```
# sysrc firewall_script="/etc/ipfw.rules"
```

To enable logging through syslogd(8), include this line:

```
# sysrc firewall_logging="YES"
```

Only firewall rules with the `log` option will be logged. The default rules do not include this option and it must be manually added. Therefore it is advisable that the default ruleset is edited for logging. In addition, log rotation may be desired if the logs are stored in a separate file.

There is no /etc/rc.conf variable to set logging limits. To limit the number of times a rule is logged per connection attempt, specify the number using this line in /etc/sysctl.conf:

```
# echo "net.inet.ip.fw.verbose_limit=5" >> /etc/sysctl.conf
```

To enable logging through a dedicated interface named `ipfw0`, add this line to /etc/rc.conf instead:

```
# sysrc firewall_logif="YES"
```

Then use tcpdump to see what is being logged:

```
# tcpdump -t -n -i ipfw0
```

There is no overhead due to logging unless tcpdump is attached.

After saving the needed edits, start the firewall. To enable logging limits now, also set the `sysctl` value specified above:

```
# service ipfw start
# sysctl net.inet.ip.fw.verbose_limit=5
```

34.4.2. IPFW Rule Syntax

When a packet enters the IPFW firewall, it is compared against the first rule in the ruleset and progresses one rule at a time, moving from top to bottom in sequence. When the packet matches the selection parameters of a rule, the rule's action is executed and the search of the ruleset terminates for that packet. This is referred to as "first match wins". If the packet does not match any of the rules, it gets caught by the mandatory IPFW default rule number 65535, which denies all packets and silently discards them. However, if the packet matches a rule that contains the `count`, `skipto`, or `tee` keywords, the search continues. Refer to ipfw(8) for details on how these keywords affect rule processing.

When creating an IPFW rule, keywords must be written in the following order. Some keywords are mandatory while other keywords are optional. The words shown in uppercase represent a variable and the words shown in lowercase must precede the variable that follows it. The `#` symbol is used to mark the start of a comment and may appear at the end of a rule or on its own line. Blank lines are ignored.

```
CMD RULE_NUMBER set SET_NUMBER ACTION log LOG_AMOUNT PROTO from SRC SRC_PORT to DST DST_PORT
OPTIONS
```

This section provides an overview of these keywords and their options. It is not an exhaustive list of every possible option. Refer to ipfw(8) for a complete description of the rule syntax that can be used when creating IPFW rules.

CMD

Every rule must start with `ipfw add`.

RULE_NUMBER

Each rule is associated with a number from `1` to `65534`. The number is used to indicate the order of rule processing. Multiple rules can have the same number, in which case they are applied according to the order in which they have been added.

SET_NUMBER

Each rule is associated with a set number from `0` to `31`. Sets can be individually disabled or enabled, making it possible to quickly add or delete a set of rules. If a SET_NUMBER is not specified, the rule will be added to set `0`.

ACTION

A rule can be associated with one of the following actions. The specified action will be executed when the packet matches the selection criterion of the rule.

`allow | accept | pass | permit`: these keywords are equivalent and allow packets that match the rule.

`check-state`: checks the packet against the dynamic state table. If a match is found, execute the action associated with the rule which generated this dynamic rule, otherwise move to the next rule. A `check-state` rule does not have selection criterion. If no `check-state` rule is present in the ruleset, the dynamic rules table is checked at the first `keep-state` or `limit` rule.

`count`: updates counters for all packets that match the rule. The search continues with the next rule.

`deny | drop`: either word silently discards packets that match this rule.

Additional actions are available. Refer to ipfw(8) for details.

LOG_AMOUNT

When a packet matches a rule with the `log` keyword, a message will be logged to syslogd(8) with a facility name of `SECURITY`. Logging only occurs if the number of packets logged for that particular rule does not exceed a specified LOG_AMOUNT. If no LOG_AMOUNT is specified, the limit is taken from the value of `net.inet.ip.fw.verbose_limit`. A value of zero removes the logging limit. Once the limit is reached, logging can be re-enabled by clearing the logging counter or the packet counter for that rule, using `ipfw resetlog`.

Logging is done after all other packet matching conditions have been met, and before performing the final action on the packet. The administrator decides which rules to enable logging on.

PROTO

This optional value can be used to specify any protocol name or number found in /etc/protocols.

SRC

The `from` keyword must be followed by the source address or a keyword that represents the source address. An address can be represented by `any`, `me` (any address configured on an interface on this system), `me6`, (any IPv6 address configured on an interface on this system), or `table` followed by the number of a lookup table which contains a list of addresses. When specifying an IP address, it can be optionally followed by its CIDR mask or subnet mask. For example, `1.2.3.4/25` or `1.2.3.4:255.255.255.128`.

SRC_PORT

An optional source port can be specified using the port number or name from /etc/services.

DST

The `to` keyword must be followed by the destination address or a keyword that represents the destination address. The same keywords and addresses described in the SRC section can be used to describe the destination.

DST_PORT

An optional destination port can be specified using the port number or name from /etc/services.

OPTIONS

Several keywords can follow the source and destination. As the name suggests, OPTIONS are optional. Commonly used options include `in` or `out`, which specify the direction of packet flow, `icmptypes` followed by the type of ICMP message, and `keep-state`.

When a `keep-state` rule is matched, the firewall will create a dynamic rule which matches bidirectional traffic between the source and destination addresses and ports using the same protocol.

The dynamic rules facility is vulnerable to resource depletion from a SYN-flood attack which would open a huge number of dynamic rules. To counter this type of attack with IPFW, use `limit`. This option limits the number of simultaneous sessions by checking the open dynamic rules, counting the number of times this rule and IP address combination occurred. If this count is greater than the value specified by `limit`, the packet is discarded.

Dozens of OPTIONS are available. Refer to ipfw(8) for a description of each available option.

34.4.3. Example Ruleset

This section demonstrates how to create an example stateful firewall ruleset script named /etc/ipfw.rules. In this example, all connection rules use `in` or `out` to clarify the direction. They also use `via` *interface-name* to specify the interface the packet is traveling over.

When first creating or testing a firewall ruleset, consider temporarily setting this tunable:

```
net.inet.ip.fw.default_to_accept="1"
```

This sets the default policy of ipfw(8) to be more permissive than the default `deny ip from any to any`, making it slightly more difficult to get locked out of the system right after a reboot.

The firewall script begins by indicating that it is a POSIX® shell script and flushes any existing rules. It then creates the `cmd` variable so that `ipfw add` does not have to be typed at the beginning of every rule. It also defines the `pif` variable which represents the name of the interface that is attached to the Internet.

```
#!/bin/sh
# Flush out the list before we begin.
ipfw -q -f flush

# Set rules command prefix
cmd="ipfw -q add"
pif="dc0"     # interface name of NIC attached to Internet
```

The first two rules allow all traffic on the trusted internal interface and on the loopback interface:

```
# Change xl0 to LAN NIC interface name
$cmd 00005 allow all from any to any via xl0

# No restrictions on Loopback Interface
$cmd 00010 allow all from any to any via lo0
```

The next rule allows the packet through if it matches an existing entry in the dynamic rules table:

```
$cmd 00101 check-state
```

The next set of rules defines which stateful connections internal systems can create to hosts on the Internet:

```
# Allow access to public DNS
# Replace x.x.x.x with the IP address of a public DNS server
# and repeat for each DNS server in /etc/resolv.conf
$cmd 00110 allow tcp from any to x.x.x.x 53 out via $pif setup keep-state
$cmd 00111 allow udp from any to x.x.x.x 53 out via $pif keep-state

# Allow access to ISP's DHCP server for cable/DSL configurations.
# Use the first rule and check log for IP address.
# Then, uncomment the second rule, input the IP address, and delete the first rule
$cmd 00120 allow log udp from any to any 67 out via $pif keep-state
#$cmd 00120 allow udp from any to x.x.x.x 67 out via $pif keep-state

# Allow outbound HTTP and HTTPS connections
$cmd 00200 allow tcp from any to any 80 out via $pif setup keep-state
$cmd 00220 allow tcp from any to any 443 out via $pif setup keep-state

# Allow outbound email connections
$cmd 00230 allow tcp from any to any 25 out via $pif setup keep-state
$cmd 00231 allow tcp from any to any 110 out via $pif setup keep-state

# Allow outbound ping
$cmd 00250 allow icmp from any to any out via $pif keep-state

# Allow outbound NTP
$cmd 00260 allow udp from any to any 123 out via $pif keep-state

# Allow outbound SSH
$cmd 00280 allow tcp from any to any 22 out via $pif setup keep-state

# deny and log all other outbound connections
$cmd 00299 deny log all from any to any out via $pif
```

The next set of rules controls connections from Internet hosts to the internal network. It starts by denying packets typically associated with attacks and then explicitly allows specific types of connections. All the authorized services that originate from the Internet use `limit` to prevent flooding.

```
# Deny all inbound traffic from non-routable reserved address spaces
$cmd 00300 deny all from 192.168.0.0/16 to any in via $pif     #RFC 1918 private IP
$cmd 00301 deny all from 172.16.0.0/12 to any in via $pif      #RFC 1918 private IP
$cmd 00302 deny all from 10.0.0.0/8 to any in via $pif         #RFC 1918 private IP
$cmd 00303 deny all from 127.0.0.0/8 to any in via $pif        #loopback
$cmd 00304 deny all from 0.0.0.0/8 to any in via $pif          #loopback
$cmd 00305 deny all from 169.254.0.0/16 to any in via $pif     #DHCP auto-config
$cmd 00306 deny all from 192.0.2.0/24 to any in via $pif       #reserved for docs
$cmd 00307 deny all from 204.152.64.0/23 to any in via $pif    #Sun cluster
interconnect
$cmd 00308 deny all from 224.0.0.0/3 to any in via $pif        #Class D & E multicast

# Deny public pings
$cmd 00310 deny icmp from any to any in via $pif

# Deny ident
$cmd 00315 deny tcp from any to any 113 in via $pif

# Deny all Netbios services.
$cmd 00320 deny tcp from any to any 137 in via $pif
$cmd 00321 deny tcp from any to any 138 in via $pif
```

```
$cmd 00322 deny tcp from any to any 139 in via $pif
$cmd 00323 deny tcp from any to any 81 in via $pif

# Deny fragments
$cmd 00330 deny all from any to any frag in via $pif

# Deny ACK packets that did not match the dynamic rule table
$cmd 00332 deny tcp from any to any established in via $pif

# Allow traffic from ISP's DHCP server.
# Replace x.x.x.x with the same IP address used in rule 00120.
#$cmd 00360 allow udp from any to x.x.x.x 67 in via $pif keep-state

# Allow HTTP connections to internal web server
$cmd 00400 allow tcp from any to me 80 in via $pif setup limit src-addr 2

# Allow inbound SSH connections
$cmd 00410 allow tcp from any to me 22 in via $pif setup limit src-addr 2

# Reject and log all other incoming connections
$cmd 00499 deny log all from any to any in via $pif
```

The last rule logs all packets that do not match any of the rules in the ruleset:

```
# Everything else is denied and logged
$cmd 00999 deny log all from any to any
```

34.4.4. In-kernel NAT

FreeBSD's IPFW firewall has two implementations of NAT: the userland implementation natd(8), and the more recent in-kernel NAT implementation. Both work in conjunction with IPFW to provide network address translation. This can be used to provide an Internet Connection Sharing solution so that several internal computers can connect to the Internet using a single public IP address.

To do this, the FreeBSD machine connected to the Internet must act as a gateway. This system must have two NICs, where one is connected to the Internet and the other is connected to the internal LAN. Each machine connected to the LAN should be assigned an IP address in the private network space, as defined by RFC 1918.

Some additional configuration is needed in order to enable the in-kernel NAT facility of IPFW. To enable in-kernel NAT support at boot time, the following must be set in /etc/rc.conf:

```
gateway_enable="YES"
firewall_enable="YES"
firewall_nat_enable="YES"
```

When `firewall_nat_enable` is set but `firewall_enable` is not, it will have no effect and do nothing. This is because the in-kernel NAT implementation is only compatible with IPFW.

When the ruleset contains stateful rules, the positioning of the NAT rule is critical and the `skipto` action is used. The `skipto` action requires a rule number so that it knows which rule to jump to. The example below builds upon the firewall ruleset shown in the previous section. It adds some additional entries and modifies some existing rules in order to configure the firewall for in-kernel NAT. It starts by adding some additional variables which represent the rule number to skip to, the `keep-state` option, and a list of TCP ports which will be used to reduce the number of rules.

```
#!/bin/sh
ipfw -q -f flush
cmd="ipfw -q add"
skip="skipto 1000"
pif=dc0
ks="keep-state"
good_tcpo="22,25,37,53,80,443,110"
```

With in-kernel NAT it is necessary to disable TCP segmentation offloading (TSO) due to the architecture of libalias(3), a library implemented as a kernel module to provide the in-kernel NAT facility of IPFW. TSO can be disabled on a per network interface basis using ifconfig(8) or on a system wide basis using sysctl(8). To disable TSO system wide, the following must be set it /etc/sysctl.conf:

```
net.inet.tcp.tso="0"
```

A NAT instance will also be configured. It is possible to have multiple NAT instances each with their own configuration. For this example only one NAT instance is needed, NAT instance number 1. The configuration can take a few options such as: `if` which indicates the public interface, `same_ports` which takes care that aliased ports and local port numbers are mapped the same, `unreg_only` will result in only unregistered (private) address spaces to be processed by the NAT instance, and `reset` which will help to keep a functioning NAT instance even when the public IP address of the IPFW machine changes. For all possible options that can be passed to a single NAT instance configuration consult ipfw(8). When configuring a stateful NATing firewall, it is necessary to allow translated packets to be reinjected in the firewall for further processing. This can be achieved by disabling `one_pass` behavior at the start of the firewall script.

```
ipfw disable one_pass
ipfw -q nat 1 config if $pif same_ports unreg_only reset
```

The inbound NAT rule is inserted *after* the two rules which allow all traffic on the trusted and loopback interfaces and after the reassemble rule but *before* the `check-state` rule. It is important that the rule number selected for this NAT rule, in this example `100`, is higher than the first three rules and lower than the `check-state` rule. Furthermore, because of the behavior of in-kernel NAT it is advised to place a reassemble rule just before the first NAT rule and after the rules that allow traffic on trusted interface. Normally, IP fragmentation should not happen, but when dealing with IPSEC/ESP/GRE tunneling traffic it might and the reassembling of fragments is necessary before handing the complete packet over to the in-kernel NAT facility.

> The reassemble rule was not needed with userland natd(8) because the internal workings of the IPFW `divert` action already takes care of reassembling packets before delivery to the socket as also stated in ipfw(8).
>
> The NAT instance and rule number used in this example does not match with the default NAT instance and rule number created by rc.firewall. rc.firewall is a script that sets up the default firewall rules present in FreeBSD.

```
$cmd 005 allow all from any to any via xl0  # exclude LAN traffic
$cmd 010 allow all from any to any via lo0  # exclude loopback traffic
$cmd 099 reass all from any to any in       # reassemble inbound packets
$cmd 100 nat 1 ip from any to any in via $pif # NAT any inbound packets
# Allow the packet through if it has an existing entry in the dynamic rules table
$cmd 101 check-state
```

The outbound rules are modified to replace the `allow` action with the `$skip` variable, indicating that rule processing will continue at rule `1000`. The seven `tcp` rules have been replaced by rule `125` as the `$good_tcpo` variable contains the seven allowed outbound ports.

> Remember that IPFW's performance is largely determined by the number of rules present in the ruleset.

```
# Authorized outbound packets
$cmd 120 $skip udp from any to x.x.x.x 53 out via $pif $ks
$cmd 121 $skip udp from any to x.x.x.x 67 out via $pif $ks
$cmd 125 $skip tcp from any to any $good_tcpo out via $pif setup $ks
$cmd 130 $skip icmp from any to any out via $pif $ks
```

The inbound rules remain the same, except for the very last rule which removes the `via $pif` in order to catch both inbound and outbound rules. The NAT rule must follow this last outbound rule, must have a higher number than that last rule, and the rule number must be referenced by the `skipto` action. In this ruleset, rule number `1000` handles passing all packets to our configured instance for NAT processing. The next rule allows any packet which has undergone NAT processing to pass.

```
$cmd 999 deny log all from any to any
$cmd 1000 nat 1 ip from any to any out via $pif # skipto location for outbound
stateful rules
$cmd 1001 allow ip from any to any
```

In this example, rules `100`, `101`, `125`, `1000`, and `1001` control the address translation of the outbound and inbound packets so that the entries in the dynamic state table always register the private LANIP address.

Consider an internal web browser which initializes a new outbound HTTP session over port 80. When the first outbound packet enters the firewall, it does not match rule `100` because it is headed out rather than in. It passes rule `101` because this is the first packet and it has not been posted to the dynamic state table yet. The packet finally matches rule `125` as it is outbound on an allowed port and has a source IP address from the internal LAN. On matching this rule, two actions take place. First, the `keep-state` action adds an entry to the dynamic state table and the specified action, `skipto rule 1000`, is executed. Next, the packet undergoes NAT and is sent out to the Internet. This packet makes its way to the destination web server, where a response packet is generated and sent back. This new packet enters the top of the ruleset. It matches rule `100` and has its destination IP address mapped back to the original internal address. It then is processed by the `check-state` rule, is found in the table as an existing session, and is released to the LAN.

On the inbound side, the ruleset has to deny bad packets and allow only authorized services. A packet which matches an inbound rule is posted to the dynamic state table and the packet is released to the LAN. The packet generated as a response is recognized by the `check-state` rule as belonging to an existing session. It is then sent to rule `1000` to undergo NAT before being released to the outbound interface.

Transitioning from userland natd(8) to in-kernel NAT might appear seamless at first but there is small catch. When using the GENERIC kernel, IPFW will load the libalias.ko kernel module, when `firewall_nat_enable` is enabled in /etc/rc.conf. The libalias.ko kernel module only provides basic NAT functionality, whereas the userland implementation natd(8) has all NAT functionality available in its userland library without any extra configuration. All functionality refers to the following kernel modules that can additionally be loaded when needed besides the standard libalias.ko kernel module: alias_ftp.ko, alias_bbt.ko, skinny.ko, irc.ko, alias_pptp.ko and alias_smedia.ko using the `kld_list` directive in /etc/rc.conf. If a custom kernel is used, the full functionality of the userland library can be compiled in, in the kernel, using the `options LIBALIAS`.

Port Redirection

The drawback with NAT in general is that the LAN clients are not accessible from the Internet. Clients on the LAN can make outgoing connections to the world but cannot receive incoming ones. This presents a problem if trying to run Internet services on one of the LAN client machines. A simple way around this is to redirect selected Internet ports on the NAT providing machine to a LAN client.

For example, an IRC server runs on client `A` and a web server runs on client `B`. For this to work properly, connections received on ports 6667 (IRC) and 80 (HTTP) must be redirected to the respective machines.

With in-kernel NAT all configuration is done in the NAT instance configuration. For a full list of options that an in-kernel NAT instance can use, consult ipfw(8). The IPFW syntax follows the syntax of natd. The syntax for `redirect_port` is as follows:

```
redirect_port proto targetIP:targetPORT[-targetPORT]
  [aliasIP:]aliasPORT[-aliasPORT]
  [remoteIP[:remotePORT[-remotePORT]]]
```

To configure the above example setup, the arguments should be:

```
redirect_port tcp 192.168.0.2:6667 6667
redirect_port tcp 192.168.0.3:80 80
```

After adding these arguments to the configuration of NAT instance 1 in the above ruleset, the TCP ports will be port forwarded to the LAN client machines running the IRC and HTTP services.

```
ipfw -q nat 1 config if $pif same_ports unreg_only reset \
  redirect_port tcp 192.168.0.2:6667 6667 \
  redirect_port tcp 192.168.0.3:80 80
```

Port ranges over individual ports can be indicated with `redirect_port`. For example, *tcp 192.168.0.2:2000-3000 2000-3000* would redirect all connections received on ports 2000 to 3000 to ports 2000 to 3000 on client `A`.

Address Redirection

Address redirection is useful if more than one IP address is available. Each LAN client can be assigned its own external IP address by ipfw(8), which will then rewrite outgoing packets from the LAN clients with the proper external IP address and redirects all traffic incoming on that particular IP address back to the specific LAN client. This is also known as static NAT. For example, if IP addresses `128.1.1.1`, `128.1.1.2`, and `128.1.1.3` are available, `128.1.1.1` can be used as the ipfw(8) machine's external IP address, while `128.1.1.2` and `128.1.1.3` are forwarded back to LAN clients `A` and `B`.

The `redirect_addr` syntax is as below, where `localIP` is the internal IP address of the LAN client, and `publicIP` the external IP address corresponding to the LAN client.

```
redirect_addr localIP publicIP
```

In the example, the arguments would read:

```
redirect_addr 192.168.0.2 128.1.1.2
redirect_addr 192.168.0.3 128.1.1.3
```

Like `redirect_port`, these arguments are placed in a NAT instance configuration. With address redirection, there is no need for port redirection, as all data received on a particular IP address is redirected.

The external IP addresses on the ipfw(8) machine must be active and aliased to the external interface. Refer to rc.conf(5) for details.

Userspace NAT

Let us start with a statement: the userspace NAT implementation: natd(8), has more overhead than in-kernel NAT. For natd(8) to translate packets, the packets have to be copied from the kernel to userspace and back which brings in extra overhead that is not present with in-kernel NAT.

To enable the userspace NAT daemon natd(8) at boot time, the following is a minimum configuration in /etc/rc.conf. Where `natd_interface` is set to the name of the NIC attached to the Internet. The rc(8) script of natd(8) will automatically check if a dynamic IP address is used and configure itself to handle that.

```
gateway_enable="YES"
natd_enable="YES"
natd_interface="rl0"
```

In general, the above ruleset as explained for in-kernel NAT can also be used together with natd(8). The exceptions are the configuration of the in-kernel NAT instance `(ipfw -q nat 1 config …)` which is not needed together with reassemble rule 99 because its functionality is included in the `divert` action. Rule number 100 and 1000 will have to change slightly as shown below.

```
$cmd 100 divert natd ip from any to any in via $pif
$cmd 1000 divert natd ip from any to any out via $pif
```

To configure port or address redirection, a similar syntax as with in-kernel NAT is used. Although, now, instead of specifying the configuration in our ruleset script like with in-kernel NAT, configuration of natd(8) is best done in a configuration file. To do this, an extra flag must be passed via /etc/rc.conf which specifies the path of the configuration file.

```
natd_flags="-f /etc/natd.conf"
```

The specified file must contain a list of configuration options, one per line. For more information about the configuration file and possible variables, consult natd(8). Below are two example entries, one per line:

```
redirect_port tcp 192.168.0.2:6667 6667
redirect_addr 192.168.0.3 128.1.1.3
```

34.4.5. The IPFW Command

`ipfw` can be used to make manual, single rule additions or deletions to the active firewall while it is running. The problem with using this method is that all the changes are lost when the system reboots. It is recommended to instead write all the rules in a file and to use that file to load the rules at boot time and to replace the currently running firewall rules whenever that file changes.

`ipfw` is a useful way to display the running firewall rules to the console screen. The IPFW accounting facility dynamically creates a counter for each rule that counts each packet that matches the rule. During the process of testing a rule, listing the rule with its counter is one way to determine if the rule is functioning as expected.

To list all the running rules in sequence:

```
# ipfw list
```

To list all the running rules with a time stamp of when the last time the rule was matched:

```
# ipfw -t list
```

The next example lists accounting information and the packet count for matched rules along with the rules themselves. The first column is the rule number, followed by the number of matched packets and bytes, followed by the rule itself.

```
# ipfw -a list
```

To list dynamic rules in addition to static rules:

```
# ipfw -d list
```

To also show the expired dynamic rules:

```
# ipfw -d -e list
```

To zero the counters:

```
# ipfw zero
```

To zero the counters for just the rule with number *NUM*:

```
# ipfw zero NUM
```

Logging Firewall Messages

Even with the logging facility enabled, IPFW will not generate any rule logging on its own. The firewall administrator decides which rules in the ruleset will be logged, and adds the `log` keyword to those rules. Normally only deny rules are logged. It is customary to duplicate the "ipfw default deny everything" rule with the `log` keyword included as the last rule in the ruleset. This way, it is possible to see all the packets that did not match any of the rules in the ruleset.

Logging is a two edged sword. If one is not careful, an over abundance of log data or a DoS attack can fill the disk with log files. Log messages are not only written to syslogd, but also are displayed on the root console screen and soon become annoying.

The `IPFIREWALL_VERBOSE_LIMIT=5` kernel option limits the number of consecutive messages sent to syslogd(8), concerning the packet matching of a given rule. When this option is enabled in the kernel, the number of consecutive messages concerning a particular rule is capped at the number specified. There is nothing to be gained from 200 identical log messages. With this option set to five, five consecutive messages concerning a particular rule would be logged to syslogd and the remainder identical consecutive messages would be counted and posted to syslogd with a phrase like the following:

```
last message repeated 45 times
```

All logged packets messages are written by default to /var/log/security, which is defined in /etc/syslog.conf.

Building a Rule Script

Most experienced IPFW users create a file containing the rules and code them in a manner compatible with running them as a script. The major benefit of doing this is the firewall rules can be refreshed in mass without the need of rebooting the system to activate them. This method is convenient in testing new rules as the procedure can be executed as many times as needed. Being a script, symbolic substitution can be used for frequently used values to be substituted into multiple rules.

This example script is compatible with the syntax used by the sh(1), csh(1), and tcsh(1) shells. Symbolic substitution fields are prefixed with a dollar sign ($). Symbolic fields do not have the $ prefix. The value to populate the symbolic field must be enclosed in double quotes ("").

Start the rules file like this:

```
############### start of example ipfw rules script #############
#
ipfw -q -f flush       # Delete all rules
# Set defaults
oif="tun0"             # out interface
odns="192.0.2.11"      # ISP's DNS server IP address
cmd="ipfw -q add "     # build rule prefix
ks="keep-state"        # just too lazy to key this each time
$cmd 00500 check-state
$cmd 00502 deny all from any to any frag
$cmd 00501 deny tcp from any to any established
$cmd 00600 allow tcp from any to any 80 out via $oif setup $ks
```

```
$cmd 00610 allow tcp from any to $odns 53 out via $oif setup $ks
$cmd 00611 allow udp from any to $odns 53 out via $oif $ks
################### End of example ipfw rules script ############
```

The rules are not important as the focus of this example is how the symbolic substitution fields are populated.

If the above example was in /etc/ipfw.rules, the rules could be reloaded by the following command:

```
# sh /etc/ipfw.rules
```

/etc/ipfw.rules can be located anywhere and the file can have any name.

The same thing could be accomplished by running these commands by hand:

```
# ipfw -q -f flush
# ipfw -q add check-state
# ipfw -q add deny all from any to any frag
# ipfw -q add deny tcp from any to any established
# ipfw -q add allow tcp from any to any 80 out via tun0 setup keep-state
# ipfw -q add allow tcp from any to 192.0.2.11 53 out via tun0 setup keep-state
# ipfw -q add 00611 allow udp from any to 192.0.2.11 53 out via tun0 keep-state
```

34.4.6. IPFW Kernel Options

In order to statically compile IPFW support into a custom kernel, refer to the instructions in Configuring the FreeBSD Kernel. The following options are available for the custom kernel configuration file:

```
options    IPFIREWALL           # enables IPFW
options    IPFIREWALL_VERBOSE       # enables logging for rules with log keyword to
syslogd(8)
options    IPFIREWALL_VERBOSE_LIMIT=5   # limits number of logged packets per-entry
options    IPFIREWALL_DEFAULT_TO_ACCEPT # sets default policy to pass what is not
explicitly denied
options    IPFIREWALL_NAT       # enables basic in-kernel NAT support
options    LIBALIAS         # enables full in-kernel NAT support
options    IPFIREWALL_NAT64     # enables in-kernel NAT64 support
options    IPFIREWALL_NPTV6     # enables in-kernel IPv6 NPT support
options    IPFIREWALL_PMOD      # enables protocols modification module support
options    IPDIVERT         # enables NAT through natd(8)
```

IPFW can be loaded as a kernel module: options above are built by default as modules or can be set at runtime using tunables.

34.5. IPFILTER (IPF)

IPFILTER, also known as IPF, is a cross-platform, open source firewall which has been ported to several operating systems, including FreeBSD, NetBSD, OpenBSD, and Solaris™.

IPFILTER is a kernel-side firewall and NAT mechanism that can be controlled and monitored by userland programs. Firewall rules can be set or deleted using ipf, NAT rules can be set or deleted using ipnat, run-time statistics for the kernel parts of IPFILTER can be printed using ipfstat, and ipmon can be used to log IPFILTER actions to the system log files.

IPF was originally written using a rule processing logic of "the last matching rule wins" and only used stateless rules. Since then, IPF has been enhanced to include the `quick` and `keep state` options.

The IPF FAQ is at http://www.phildev.net/ipf/index.html. A searchable archive of the IPFilter mailing list is available at http://marc.info/?l=ipfilter.

This section of the Handbook focuses on IPF as it pertains to FreeBSD. It provides examples of rules that contain the `quick` and `keep state` options.

34.5.1. Enabling IPF

IPF is included in the basic FreeBSD install as a kernel loadable module, meaning that a custom kernel is not needed in order to enable IPF.

For users who prefer to statically compile IPF support into a custom kernel, refer to the instructions in Configuring the FreeBSD Kernel. The following kernel options are available:

```
options IPFILTER
options IPFILTER_LOG
options IPFILTER_LOOKUP
options IPFILTER_DEFAULT_BLOCK
```

where `options IPFILTER` enables support for IPFILTER, `options IPFILTER_LOG` enables IPF logging using the ipl packet logging pseudo-device for every rule that has the `log` keyword, `IPFILTER_LOOKUP` enables IP pools in order to speed up IP lookups, and `options IPFILTER_DEFAULT_BLOCK` changes the default behavior so that any packet not matching a firewall `pass` rule gets blocked.

To configure the system to enable IPF at boot time, add the following entries to /etc/rc.conf. These entries will also enable logging and `default pass all`. To change the default policy to `block all` without compiling a custom kernel, remember to add a `block all` rule at the end of the ruleset.

```
ipfilter_enable="YES"             # Start ipf firewall
ipfilter_rules="/etc/ipf.rules"   # loads rules definition text file
ipv6_ipfilter_rules="/etc/ipf6.rules" # loads rules definition text file for IPv6
ipmon_enable="YES"                # Start IP monitor log
ipmon_flags="-Ds"                 # D = start as daemon
                                  # s = log to syslog
                                  # v = log tcp window, ack, seq
                                  # n = map IP & port to names
```

If NAT functionality is needed, also add these lines:

```
gateway_enable="YES"              # Enable as LAN gateway
ipnat_enable="YES"                # Start ipnat function
ipnat_rules="/etc/ipnat.rules"    # rules definition file for ipnat
```

Then, to start IPF now:

```
#  service ipfilter start
```

To load the firewall rules, specify the name of the ruleset file using `ipf`. The following command can be used to replace the currently running firewall rules:

```
# ipf -Fa -f /etc/ipf.rules
```

where `-Fa` flushes all the internal rules tables and `-f` specifies the file containing the rules to load.

This provides the ability to make changes to a custom ruleset and update the running firewall with a fresh copy of the rules without having to reboot the system. This method is convenient for testing new rules as the procedure can be executed as many times as needed.

Refer to ipf(8) for details on the other flags available with this command.

34.5.2. IPF Rule Syntax

This section describes the IPF rule syntax used to create stateful rules. When creating rules, keep in mind that unless the `quick` keyword appears in a rule, every rule is read in order, with the *last matching rule* being the one that is applied. This means that even if the first rule to match a packet is a `pass`, if there is a later matching rule that is a `block`, the packet will be dropped. Sample rulesets can be found in /usr/share/examples/ipfilter.

When creating rules, a `#` character is used to mark the start of a comment and may appear at the end of a rule, to explain that rule's function, or on its own line. Any blank lines are ignored.

The keywords which are used in rules must be written in a specific order, from left to right. Some keywords are mandatory while others are optional. Some keywords have sub-options which may be keywords themselves and also include more sub-options. The keyword order is as follows, where the words shown in uppercase represent a variable and the words shown in lowercase must precede the variable that follows it:

`ACTION DIRECTION OPTIONS proto PROTO_TYPE from SRC_ADDR SRC_PORT to DST_ADDR DST_PORT TCP_FLAG|ICMP_TYPE keep state STATE`

This section describes each of these keywords and their options. It is not an exhaustive list of every possible option. Refer to ipf(5) for a complete description of the rule syntax that can be used when creating IPF rules and examples for using each keyword.

ACTION

The action keyword indicates what to do with the packet if it matches that rule. Every rule *must* have an action. The following actions are recognized:

`block`: drops the packet.

`pass`: allows the packet.

`log`: generates a log record.

`count`: counts the number of packets and bytes which can provide an indication of how often a rule is used.

`auth`: queues the packet for further processing by another program.

`call`: provides access to functions built into IPF that allow more complex actions.

`decapsulate`: removes any headers in order to process the contents of the packet.

DIRECTION

Next, each rule must explicitly state the direction of traffic using one of these keywords:

`in`: the rule is applied against an inbound packet.

`out`: the rule is applied against an outbound packet.

`all`: the rule applies to either direction.

If the system has multiple interfaces, the interface can be specified along with the direction. An example would be `in on fxp0`.

OPTIONS

Options are optional. However, if multiple options are specified, they must be used in the order shown here.

`log`: when performing the specified ACTION, the contents of the packet's headers will be written to the ipl(4) packet log pseudo-device.

`quick`: if a packet matches this rule, the ACTION specified by the rule occurs and no further processing of any following rules will occur for this packet.

`on`: must be followed by the interface name as displayed by ifconfig(8). The rule will only match if the packet is going through the specified interface in the specified direction.

When using the `log` keyword, the following qualifiers may be used in this order:

`body`: indicates that the first 128 bytes of the packet contents will be logged after the headers.

`first`: if the `log` keyword is being used in conjunction with a `keep state` option, this option is recommended so that only the triggering packet is logged and not every packet which matches the stateful connection.

Additional options are available to specify error return messages. Refer to ipf(5) for more details.

PROTO_TYPE

The protocol type is optional. However, it is mandatory if the rule needs to specify a SRC_PORT or a DST_PORT as it defines the type of protocol. When specifying the type of protocol, use the `proto` keyword followed by either a protocol number or name from /etc/protocols. Example protocol names include `tcp`, `udp`, or `icmp`. If PROTO_TYPE is specified but no SRC_PORT or DST_PORT is specified, all port numbers for that protocol will match that rule.

SRC_ADDR

The `from` keyword is mandatory and is followed by a keyword which represents the source of the packet. The source can be a hostname, an IP address followed by the CIDR mask, an address pool, or the keyword `all`. Refer to ipf(5) for examples.

There is no way to match ranges of IP addresses which do not express themselves easily using the dotted numeric form / mask-length notation. The net-mgmt/ipcalc package or port may be used to ease the calculation of the CIDR mask. Additional information is available at the utility's web page: http://jodies.de/ipcalc.

SRC_PORT

The port number of the source is optional. However, if it is used, it requires PROTO_TYPE to be first defined in the rule. The port number must also be preceded by the `proto` keyword.

A number of different comparison operators are supported: `=` (equal to), `!=` (not equal to), `<` (less than), `>` (greater than), `⇐` (less than or equal to), and `>=` (greater than or equal to).

To specify port ranges, place the two port numbers between `<>` (less than and greater than), `><` (greater than and less than), or `:` (greater than or equal to and less than or equal to).

DST_ADDR

The `to` keyword is mandatory and is followed by a keyword which represents the destination of the packet. Similar to SRC_ADDR, it can be a hostname, an IP address followed by the CIDR mask, an address pool, or the keyword `all`.

DST_PORT

Similar to SRC_PORT, the port number of the destination is optional. However, if it is used, it requires PROTO_TYPE to be first defined in the rule. The port number must also be preceded by the `proto` keyword.

TCP_FLAG|ICMP_TYPE

If `tcp` is specified as the PROTO_TYPE, flags can be specified as letters, where each letter represents one of the possible TCP flags used to determine the state of a connection. Possible values are: `S` (SYN), `A` (ACK), `P` (PSH), `F` (FIN), `U` (URG), `R` (RST), `C` (CWN), and `E` (ECN).

If `icmp` is specified as the PROTO_TYPE, the ICMP type to match can be specified. Refer to ipf(5) for the allowable types.

STATE

If a `pass` rule contains `keep state`, IPF will add an entry to its dynamic state table and allow subsequent packets that match the connection. IPF can track state for TCP, UDP, and ICMP sessions. Any packet that IPF can be certain is part of an active session, even if it is a different protocol, will be allowed.

In IPF, packets destined to go out through the interface connected to the public Internet are first checked against the dynamic state table. If the packet matches the next expected packet comprising an active session conversation, it exits the firewall and the state of the session conversation flow is updated in the dynamic state table. Packets that do not belong to an already active session are checked against the outbound ruleset. Packets coming in from the interface connected to the public Internet are first checked against the dynamic state table. If the packet matches the next expected packet comprising an active session, it exits the firewall and the state of the session conversation flow is updated in the dynamic state table. Packets that do not belong to an already active session are checked against the inbound ruleset.

Several keywords can be added after `keep state`. If used, these keywords set various options that control stateful filtering, such as setting connection limits or connection age. Refer to ipf(5) for the list of available options and their descriptions.

34.5.3. Example Ruleset

This section demonstrates how to create an example ruleset which only allows services matching `pass` rules and blocks all others.

FreeBSD uses the loopback interface (lo0) and the IP address `127.0.0.1` for internal communication. The firewall ruleset must contain rules to allow free movement of these internally used packets:

```
# no restrictions on loopback interface
pass in quick on lo0 all
pass out quick on lo0 all
```

The public interface connected to the Internet is used to authorize and control access of all outbound and inbound connections. If one or more interfaces are cabled to private networks, those internal interfaces may require rules to allow packets originating from the LAN to flow between the internal networks or to the interface attached to the Internet. The ruleset should be organized into three major sections: any trusted internal interfaces, outbound connections through the public interface, and inbound connections through the public interface.

These two rules allow all traffic to pass through a trusted LAN interface named xl0:

```
# no restrictions on inside LAN interface for private network
pass out quick on xl0 all
pass in quick on xl0 all
```

The rules for the public interface's outbound and inbound sections should have the most frequently matched rules placed before less commonly matched rules, with the last rule in the section blocking and logging all packets for that interface and direction.

This set of rules defines the outbound section of the public interface named dc0. These rules keep state and identify the specific services that internal systems are authorized for public Internet access. All the rules use `quick` and specify the appropriate port numbers and, where applicable, destination addresses.

```
# interface facing Internet (outbound)
# Matches session start requests originating from or behind the
# firewall, destined for the Internet.

# Allow outbound access to public DNS servers.
# Replace x.x.x.x with address listed in /etc/resolv.conf.
# Repeat for each DNS server.
pass out quick on dc0 proto tcp from any to x.x.x.x port = 53 flags S keep state
pass out quick on dc0 proto udp from any to x.x.x.x port = 53 keep state

# Allow access to ISP's specified DHCP server for cable or DSL networks.
# Use the first rule, then check log for the IP address of DHCP server.
# Then, uncomment the second rule, replace z.z.z.z with the IP address,
# and comment out the first rule
pass out log quick on dc0 proto udp from any to any port = 67 keep state
#pass out quick on dc0 proto udp from any to z.z.z.z port = 67 keep state

# Allow HTTP and HTTPS
pass out quick on dc0 proto tcp from any to any port = 80 flags S keep state
pass out quick on dc0 proto tcp from any to any port = 443 flags S keep state

# Allow email
pass out quick on dc0 proto tcp from any to any port = 110 flags S keep state
pass out quick on dc0 proto tcp from any to any port = 25 flags S keep state

# Allow NTP
pass out quick on dc0 proto tcp from any to any port = 37 flags S keep state

# Allow FTP
pass out quick on dc0 proto tcp from any to any port = 21 flags S keep state

# Allow SSH
pass out quick on dc0 proto tcp from any to any port = 22 flags S keep state

# Allow ping
pass out quick on dc0 proto icmp from any to any icmp-type 8 keep state

# Block and log everything else
block out log first quick on dc0 all
```

This example of the rules in the inbound section of the public interface blocks all undesirable packets first. This reduces the number of packets that are logged by the last rule.

```
# interface facing Internet (inbound)
# Block all inbound traffic from non-routable or reserved address spaces
block in quick on dc0 from 192.168.0.0/16 to any    #RFC 1918 private IP
block in quick on dc0 from 172.16.0.0/12 to any     #RFC 1918 private IP
block in quick on dc0 from 10.0.0.0/8 to any        #RFC 1918 private IP
block in quick on dc0 from 127.0.0.0/8 to any       #loopback
block in quick on dc0 from 0.0.0.0/8 to any         #loopback
block in quick on dc0 from 169.254.0.0/16 to any    #DHCP auto-config
block in quick on dc0 from 192.0.2.0/24 to any      #reserved for docs
block in quick on dc0 from 204.152.64.0/23 to any   #Sun cluster interconnect
block in quick on dc0 from 224.0.0.0/3 to any       #Class D & E multicast

# Block fragments and too short tcp packets
block in quick on dc0 all with frags
block in quick on dc0 proto tcp all with short

# block source routed packets
block in quick on dc0 all with opt lsrr
block in quick on dc0 all with opt ssrr

# Block OS fingerprint attempts and log first occurrence
block in log first quick on dc0 proto tcp from any to any flags FUP

# Block anything with special options
block in quick on dc0 all with ipopts

# Block public pings and ident
block in quick on dc0 proto icmp all icmp-type 8
block in quick on dc0 proto tcp from any to any port = 113

# Block incoming Netbios services
block in log first quick on dc0 proto tcp/udp from any to any port = 137
block in log first quick on dc0 proto tcp/udp from any to any port = 138
block in log first quick on dc0 proto tcp/udp from any to any port = 139
block in log first quick on dc0 proto tcp/udp from any to any port = 81
```

Any time there are logged messages on a rule with the `log first` option, run `ipfstat -hio` to evaluate how many times the rule has been matched. A large number of matches may indicate that the system is under attack.

The rest of the rules in the inbound section define which connections are allowed to be initiated from the Internet. The last rule denies all connections which were not explicitly allowed by previous rules in this section.

```
# Allow traffic in from ISP's DHCP server. Replace z.z.z.z with
# the same IP address used in the outbound section.
pass in quick on dc0 proto udp from z.z.z.z to any port = 68 keep state

# Allow public connections to specified internal web server
```

```
pass in quick on dc0 proto tcp from any to x.x.x.x port = 80 flags S keep state

# Block and log only first occurrence of all remaining traffic.
block in log first quick on dc0 all
```

34.5.4. Configuring NAT

To enable NAT, add these statements to /etc/rc.conf and specify the name of the file containing the NAT rules:

```
gateway_enable="YES"
ipnat_enable="YES"
ipnat_rules="/etc/ipnat.rules"
```

NAT rules are flexible and can accomplish many different things to fit the needs of both commercial and home users. The rule syntax presented here has been simplified to demonstrate common usage. For a complete rule syntax description, refer to ipnat(5).

The basic syntax for a NAT rule is as follows, where `map` starts the rule and *IF* should be replaced with the name of the external interface:

```
map IF LAN_IP_RANGE -> PUBLIC_ADDRESS
```

The *LAN_IP_RANGE* is the range of IP addresses used by internal clients. Usually, it is a private address range such as `192.168.1.0/24`. The *PUBLIC_ADDRESS* can either be the static external IP address or the keyword `0/32` which represents the IP address assigned to *IF*.

In IPF, when a packet arrives at the firewall from the LAN with a public destination, it first passes through the outbound rules of the firewall ruleset. Then, the packet is passed to the NAT ruleset which is read from the top down, where the first matching rule wins. IPF tests each NAT rule against the packet's interface name and source IP address. When a packet's interface name matches a NAT rule, the packet's source IP address in the private LAN is checked to see if it falls within the IP address range specified in *LAN_IP_RANGE*. On a match, the packet has its source IP address rewritten with the public IP address specified by *PUBLIC_ADDRESS*. IPF posts an entry in its internal NAT table so that when the packet returns from the Internet, it can be mapped back to its original private IP address before being passed to the firewall rules for further processing.

For networks that have large numbers of internal systems or multiple subnets, the process of funneling every private IP address into a single public IP address becomes a resource problem. Two methods are available to relieve this issue.

The first method is to assign a range of ports to use as source ports. By adding the `portmap` keyword, NAT can be directed to only use source ports in the specified range:

```
map dc0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:60000
```

Alternately, use the `auto` keyword which tells NAT to determine the ports that are available for use:

```
map dc0 192.168.1.0/24 -> 0/32 portmap tcp/udp auto
```

The second method is to use a pool of public addresses. This is useful when there are too many LAN addresses to fit into a single public address and a block of public IP addresses is available. These public addresses can be used as a pool from which NAT selects an IP address as a packet's address is mapped on its way out.

The range of public IP addresses can be specified using a netmask or CIDR notation. These two rules are equivalent:

```
map dc0 192.168.1.0/24 -> 204.134.75.0/255.255.255.0
map dc0 192.168.1.0/24 -> 204.134.75.0/24
```

A common practice is to have a publicly accessible web server or mail server segregated to an internal network segment. The traffic from these servers still has to undergo NAT, but port redirection is needed to direct inbound traffic to the correct server. For example, to map a web server using the internal address `10.0.10.25` to its public IP address of `20.20.20.5`, use this rule:

```
rdr dc0 20.20.20.5/32 port 80 -> 10.0.10.25 port 80
```

If it is the only web server, this rule would also work as it redirects all external HTTP requests to `10.0.10.25`:

```
rdr dc0 0.0.0.0/0 port 80 -> 10.0.10.25 port 80
```

IPF has a built in FTP proxy which can be used with NAT. It monitors all outbound traffic for active or passive FTP connection requests and dynamically creates temporary filter rules containing the port number used by the FTP data channel. This eliminates the need to open large ranges of high order ports for FTP connections.

In this example, the first rule calls the proxy for outbound FTP traffic from the internal LAN. The second rule passes the FTP traffic from the firewall to the Internet, and the third rule handles all non-FTP traffic from the internal LAN:

```
map dc0 10.0.10.0/29 -> 0/32 proxy port 21 ftp/tcp
map dc0 0.0.0.0/0 -> 0/32 proxy port 21 ftp/tcp
map dc0 10.0.10.0/29 -> 0/32
```

The FTP `map` rules go before the NAT rule so that when a packet matches an FTP rule, the FTP proxy creates temporary filter rules to let the FTP session packets pass and undergo NAT. All LAN packets that are not FTP will not match the FTP rules but will undergo NAT if they match the third rule.

Without the FTP proxy, the following firewall rules would instead be needed. Note that without the proxy, all ports above `1024` need to be allowed:

```
# Allow out LAN PC client FTP to public Internet
# Active and passive modes
pass out quick on rl0 proto tcp from any to any port = 21 flags S keep state

# Allow out passive mode data channel high order port numbers
pass out quick on rl0 proto tcp from any to any port > 1024 flags S keep state

# Active mode let data channel in from FTP server
pass in quick on rl0 proto tcp from any to any port = 20 flags S keep state
```

Whenever the file containing the NAT rules is edited, run `ipnat` with `-CF` to delete the current NAT rules and flush the contents of the dynamic translation table. Include `-f` and specify the name of the NAT ruleset to load:

```
# ipnat -CF -f /etc/ipnat.rules
```

To display the NAT statistics:

```
# ipnat -s
```

To list the NAT table's current mappings:

```
# ipnat -l
```

To turn verbose mode on and display information relating to rule processing and active rules and table entries:

```
# ipnat -v
```

34.5.5. Viewing IPF Statistics

IPF includes ipfstat(8) which can be used to retrieve and display statistics which are gathered as packets match rules as they go through the firewall. Statistics are accumulated since the firewall was last started or since the last time they were reset to zero using `ipf -Z`.

The default `ipfstat` output looks like this:

```
input packets: blocked 99286 passed 1255609 nomatch 14686 counted 0
 output packets: blocked 4200 passed 1284345 nomatch 14687 counted 0
 input packets logged: blocked 99286 passed 0
 output packets logged: blocked 0 passed 0
 packets logged: input 0 output 0
 log failures: input 3898 output 0
 fragment state(in): kept 0 lost 0
 fragment state(out): kept 0 lost 0
 packet state(in): kept 169364 lost 0
 packet state(out): kept 431395 lost 0
 ICMP replies: 0 TCP RSTs sent: 0
 Result cache hits(in): 1215208 (out): 1098963
 IN Pullups succeeded: 2 failed: 0
 OUT Pullups succeeded: 0 failed: 0
 Fastroute successes: 0 failures: 0
 TCP cksum fails(in): 0 (out): 0
 Packet log flags set: (0)
```

Several options are available. When supplied with either `-i` for inbound or `-o` for outbound, the command will retrieve and display the appropriate list of filter rules currently installed and in use by the kernel. To also see the rule numbers, include `-n`. For example, `ipfstat -on` displays the outbound rules table with rule numbers:

```
@1 pass out on xl0 from any to any
@2 block out on dc0 from any to any
@3 pass out quick on dc0 proto tcp/udp from any to any keep state
```

Include `-h` to prefix each rule with a count of how many times the rule was matched. For example, `ipfstat -oh` displays the outbound internal rules table, prefixing each rule with its usage count:

```
2451423 pass out on xl0 from any to any
354727 block out on dc0 from any to any
430918 pass out quick on dc0 proto tcp/udp from any to any keep state
```

To display the state table in a format similar to top(1), use `ipfstat -t`. When the firewall is under attack, this option provides the ability to identify and see the attacking packets. The optional sub-flags give the ability to select the destination or source IP, port, or protocol to be monitored in real time. Refer to ipfstat(8) for details.

34.5.6. IPF Logging

IPF provides `ipmon`, which can be used to write the firewall's logging information in a human readable format. It requires that `options IPFILTER_LOG` be first added to a custom kernel using the instructions in Configuring the FreeBSD Kernel.

This command is typically run in daemon mode in order to provide a continuous system log file so that logging of past events may be reviewed. Since FreeBSD has a built in syslogd(8) facility to automatically rotate system logs, the default rc.conf `ipmon_flags` statement uses `-Ds`:

```
ipmon_flags="-Ds" # D = start as daemon
                  # s = log to syslog
```

```
                    # v = log tcp window, ack, seq
                    # n = map IP & port to names
```

Logging provides the ability to review, after the fact, information such as which packets were dropped, what addresses they came from, and where they were going. This information is useful in tracking down attackers.

Once the logging facility is enabled in rc.conf and started with `service ipmon start`, IPF will only log the rules which contain the `log` keyword. The firewall administrator decides which rules in the ruleset should be logged and normally only deny rules are logged. It is customary to include the `log` keyword in the last rule in the ruleset. This makes it possible to see all the packets that did not match any of the rules in the ruleset.

By default, `ipmon -Ds` mode uses `local0` as the logging facility. The following logging levels can be used to further segregate the logged data:

```
LOG_INFO - packets logged using the "log" keyword as the action rather than pass or
block.
LOG_NOTICE - packets logged which are also passed
LOG_WARNING - packets logged which are also blocked
LOG_ERR - packets which have been logged and which can be considered short due to an
incomplete header
```

In order to setup IPF to log all data to /var/log/ipfilter.log, first create the empty file:

```
# touch /var/log/ipfilter.log
```

Then, to write all logged messages to the specified file, add the following statement to /etc/syslog.conf:

```
local0.* /var/log/ipfilter.log
```

To activate the changes and instruct syslogd(8) to read the modified /etc/syslog.conf, run `service syslogd reload`.

Do not forget to edit /etc/newsyslog.conf to rotate the new log file.

Messages generated by `ipmon` consist of data fields separated by white space. Fields common to all messages are:

1. The date of packet receipt.
2. The time of packet receipt. This is in the form HH:MM:SS.F, for hours, minutes, seconds, and fractions of a second.
3. The name of the interface that processed the packet.
4. The group and rule number of the rule in the format `@0:17`.

5. The action: `p` for passed, `b` for blocked, `S` for a short packet, `n` did not match any rules, and `L` for a log rule.
6. The addresses written as three fields: the source address and port separated by a comma, the → symbol, and the destination address and port. For example: `209.53.17.22,80 → 198.73.220.17,1722`.
7. `PR` followed by the protocol name or number: for example, `PR tcp`.
8. `len` followed by the header length and total length of the packet: for example, `len 20 40`.

If the packet is a TCP packet, there will be an additional field starting with a hyphen followed by letters corresponding to any flags that were set. Refer to ipf(5) for a list of letters and their flags.

If the packet is an ICMP packet, there will be two fields at the end: the first always being "icmp" and the next being the ICMP message and sub-message type, separated by a slash. For example: `icmp 3/3` for a port unreachable message.

34.6. Blocklistd

Blocklistd is a daemon listening to sockets awaiting to receive notifications from other daemons about connection attempts that failed or were successful. It is most widely used in blocking too many connection attempts on open ports. A prime example is SSH running on the internet getting a lot of requests from bots or scripts trying to guess passwords and gain access. Using blocklistd, the daemon can notify the firewall to create a filter rule to block excessive connection attempts from a single source after a number of tries. Blocklistd was first developed on NetBSD and appeared there in version 7, it was originally called blacklistd. FreeBSD 11 imported blocklistd from NetBSD.

This chapter describes how to set up blocklistd, configure it, and provides examples on how to use it. Readers should be familiar with basic firewall concepts like rules. For details, refer to the firewall chapter. PF is used in the examples, but other firewalls available on FreeBSD should be able to work with blocklistd, too.

34.6.1. Enabling Blocklistd

The main configuration for blocklistd is stored in blocklistd.conf(5). Various command line options are also available to change blocklistd's run-time behavior. Persistent configuration across reboots should be stored in /etc/blocklistd.conf. To enable the daemon during system boot, add a `blocklistd_enable` line to /etc/rc.conf like this:

```
# sysrc blocklistd_enable=yes
```

To start the service manually, run this command:

```
# service blocklistd start
```

34.6.2. Creating a Blocklistd Ruleset

Rules for blocklistd are configured in blocklistd.conf(5) with one entry per line. Each rule contains a tuple separated by spaces or tabs. Rules either belong to a `local` or a `remote`, which applies to the machine where blocklistd is running or an outside source, respectively.

Local Rules

An example blocklistd.conf entry for a local rule looks like this:

```
[local]
ssh             stream  *       *               *       3       24h
```

All rules that follow the `[local]` section are treated as local rules (which is the default), applying to the local machine. When a `[remote]` section is encountered, all rules that follow it are handled as remote machine rules.

Seven fields separated by either tabs or spaces define a rule. The first four fields identify the traffic that should be blocklisted. The three fields that follow define blocklistd's behavior. Wildcards are denoted as asterisks (`*`), matching anything in this field. The first field defines the location. In local rules, these are the network ports. The syntax for the location field is as follows:

```
[address|interface][/mask][:port]
```

Addresses can be specified as IPv4 in numeric format or IPv6 in square brackets. An interface name like `em0` can also be used.

The socket type is defined by the second field. TCP sockets are of type `stream`, whereas UDP is denoted as `dgram`. The example above uses TCP, since SSH is using that protocol.

A protocol can be used in the third field of a blocklistd rule. The following protocols can be used: `tcp`, `udp`, `tcp6`, `udp6`, or numeric. A wildcard, like in the example, is typically used to match all protocols unless there is a reason to distinguish traffic by a certain protocol.

In the fourth field, the effective user or owner of the daemon process that is reporting the event is defined. The username or UID can be used here, as well as a wildcard (see example rule above).

The packet filter rule name is declared by the fifth field, which starts the behavior part of the rule. By default, blocklistd puts all blocks under a pf anchor called `blocklistd` in pf.conf like this:

```
anchor "blocklistd/*" in on $ext_if
block in
pass out
```

For separate blocklists, an anchor name can be used in this field. In other cases, the wildcard will suffice. When a name starts with a hyphen (`-`) it means that an anchor with the default rule name prepended should be used. A modified example from the above using the hyphen would look like this:

```
ssh             stream  *       *               -ssh       3       24h
```

With such a rule, any new blocklist rules are added to an anchor called `blocklistd-ssh`.

To block whole subnets for a single rule violation, a `/` in the rule name can be used. This causes the remaining portion of the name to be interpreted as the mask to be applied to the address specified in the rule. For example, this rule would block every address adjoining `/24`.

```
22              stream  tcp       *               */24    3       24h
```

> It is important to specify the proper protocol here. IPv4 and IPv6 treat /24 differently, that is the reason why `*` cannot be used in the third field for this rule.

This rule defines that if any one host in that network is misbehaving, everything else on that network will be blocked, too.

The sixth field, called `nfail`, sets the number of login failures required to blocklist the remote IP in question. When a wildcard is used at this position, it means that blocks will never happen. In the example rule above, a limit of three is defined meaning that after three attempts to log into SSH on one connection, the IP is blocked.

The last field in a blocklistd rule definition specifies how long a host is blocklisted. The default unit is seconds, but suffixes like `m`, `h`, and `d` can also be specified for minutes, hours, and days, respectively.

The example rule in its entirety means that after three times authenticating to SSH will result in a new PF block rule for that host. Rule matches are performed by first checking local rules one after another, from most specific to least specific. When a match occurs, the `remote` rules are applied and the `name`, `nfail`, and `duration` fields are changed by the `remote` rule that matched.

Remote Rules

Remote rules are used to specify how blocklistd changes its behavior depending on the remote host currently being evaluated. Each field in a remote rule is the same as in a local rule. The only difference is in the way blocklistd is using them. To explain it, this example rule is used:

```
[remote]
203.0.113.128/25 *      *       *               =/25    =       48h
```

The address field can be an IP address (either v4 or v6), a port or both. This allows setting special rules for a specific remote address range like in this example. The fields for socket type, protocol and owner are identically interpreted as in the local rule.

The name fields is different though: the equal sign (`=`) in a remote rule tells blocklistd to use the value from the matching local rule. It means that the firewall rule entry is taken and the `/25` prefix (a netmask of `255.255.255.128`) is added. When a connection from that address range is blocklisted, the entire subnet is affected. A PF anchor name can also be used here, in which case blocklistd will add rules for this address block to the anchor of that name. The default table is used when a wildcard is specified.

A custom number of failures in the `nfail` column can be defined for an address. This is useful for exceptions to a specific rule, to maybe allow someone a less strict application of rules or a bit more leniency in login tries. Blocking is disabled when an asterisk is used in this sixth field.

Remote rules allow a stricter enforcement of limits on attempts to log in compared to attempts coming from a local network like an office.

34.6.3. Blocklistd Client Configuration

There are a few software packages in FreeBSD that can utilize blocklistd's functionality. The two most prominent ones are ftp/freebsd-ftpd and sshd(8) to block excessive connection attempts. To activate blocklistd in the SSH daemon, add the following line to /etc/ssh/sshd_config:

```
UseBlocklist yes
```

Restart sshd afterwards to make these changes take effect.

Blocklisting for ftp/freebsd-ftpd is enabled using `-B`, either in /etc/inetd.conf or as a flag in /etc/rc.conf like this:

```
ftpd_flags="-B"
```

That is all that is needed to make these programs talk to blocklistd.

34.6.4. Blocklistd Management

Blocklistd provides the user with a management utility called blocklistctl(8). It displays blocked addresses and networks that are blocklisted by the rules defined in blocklistd.conf(5). To see the list of currently blocked hosts, use `dump` combined with `-b` like this.

```
# blocklistctl dump -b
rulename                 address/ma:port id      nfail   last access
blocklistd        213.0.123.128/25:22   OK      6/3     2019/06/08 14:30:19
```

This example shows that there were 6 out of three permitted attempts on port 22 coming from the address range `213.0.123.128/25`. There are more attempts listed than are allowed because SSH allows a client to try multiple logins on a single TCP connection. A connection that is currently going on is not stopped by blocklistd. The last connection attempt is listed in the `last access` column of the output.

To see the remaining time that this host will be on the blocklist, add `-r` to the previous command.

```
# blocklistctl dump -br
rulename                address/ma:port id      nfail   remaining time
blocklistd        213.0.123.128/25:22   OK      6/3     36s
```

In this example, there are 36s seconds left until this host will not be blocked any more.

34.6.5. Removing Hosts from the Block List

Sometimes it is necessary to remove a host from the block list before the remaining time expires. Unfortunately, there is no functionality in blocklistd to do that. However, it is possible to remove the address from the PF table using pfctl. For each blocked port, there is a child anchor inside the blocklistd anchor defined in /etc/pf.conf. For example, if there is a child anchor for blocking port 22 it is called `blocklistd/22`. There is a table inside that child anchor that contains the blocked addresses. This table is called port followed by the port number. In this example, it would be called `port22`. With that information at hand, it is now possible to use pfctl(8) to display all addresses listed like this:

```
# pfctl -a blocklistd/22 -t port22 -T show
...
213.0.123.128/25
...
```

After identifying the address to be unblocked from the list, the following command removes it from the list:

```
# pfctl -a blocklistd/22 -t port22 -T delete 213.0.123.128/25
```

The address is now removed from PF, but will still show up in the blocklistctl list, since it does not know about any changes made in PF. The entry in blocklistd's database will eventually expire and be removed from its output. The entry will be added again if the host is matching one of the block rules in blocklistd again.

Chapter 35. Advanced Networking

35.1. Synopsis

This chapter covers a number of advanced networking topics.

Read this chapter to learn:

- The basics of gateways and routes.
- How to set up USB tethering.
- How to set up IEEE® 802.11 and Bluetooth® devices.
- How to make FreeBSD act as a bridge.
- How to set up network PXE booting.
- How to enable and utilize the features of the Common Address Redundancy Protocol (CARP) in FreeBSD.
- How to configure multiple VLANs on FreeBSD.
- How to configure a bluetooth headset.

Before reading this chapter:

- Understand the basics of the /etc/rc scripts.
- Be familiar with basic network terminology.
- Understand basic network configuration on FreeBSD (FreeBSD network).
- Know how to configure and install a new FreeBSD kernel (Configuring the FreeBSD Kernel).
- Know how to install additional third-party software (Installing Applications: Packages and Ports).

35.2. Gateways and Routes

Routing is the mechanism that allows a system to find the network path to another system. A *route* is a defined pair of addresses which represent the "destination" and a "gateway". The route indicates that when trying to get to the specified destination, send the packets through the specified gateway. There are three types of destinations: individual hosts, subnets, and "default". The "default route" is used if no other routes apply. There are also three types of gateways: individual hosts, interfaces, also called links, and Ethernet hardware (MAC) addresses. Known routes are stored in a routing table.

This section provides an overview of routing basics. It then demonstrates how to configure a FreeBSD system as a router and offers some troubleshooting tips.

35.2.1. Routing Basics

To view the routing table of a FreeBSD system, use netstat(1):

```
% netstat -r
Routing tables

Internet:
Destination      Gateway            Flags     Refs      Use     Netif Expire
default          outside-gw         UGS         37      418       em0
localhost        localhost          UH           0      181       lo0
test0            0:e0:b5:36:cf:4f   UHLW         5    63288       re0     77
10.20.30.255     link#1             UHLW         1     2421
example.com      link#1             UC           0        0
host1            0:e0:a8:37:8:1e    UHLW         3     4601       lo0
host2            0:e0:a8:37:8:1e    UHLW         0        5       lo0 =>
host2.example.com link#1            UC           0        0
224              link#1             UC           0        0
```

The entries in this example are as follows:

default

The first route in this table specifies the `default` route. When the local system needs to make a connection to a remote host, it checks the routing table to determine if a known path exists. If the remote host matches an entry in the table, the system checks to see if it can connect using the interface specified in that entry.

If the destination does not match an entry, or if all known paths fail, the system uses the entry for the default route. For hosts on a local area network, the `Gateway` field in the default route is set to the system which has a direct connection to the Internet. When reading this entry, verify that the `Flags` column indicates that the gateway is usable (`UG`).

The default route for a machine which itself is functioning as the gateway to the outside world will be the gateway machine at the Internet Service Provider (ISP).

localhost

The second route is the `localhost` route. The interface specified in the `Netif` column for `localhost` is lo0, also known as the loopback device. This indicates that all traffic for this destination should be internal, rather than sending it out over the network.

MAC address

The addresses beginning with `0:e0:` are MAC addresses. FreeBSD will automatically identify any hosts, `test0` in the example, on the local Ethernet and add a route for that host over the Ethernet interface, re0. This type of route has a timeout, seen in the `Expire` column, which is used if the host does not respond in a specific amount of time. When this happens, the route to this host will be automatically deleted. These hosts are identified using the Routing Information Protocol (RIP), which calculates routes to local hosts based upon a shortest path determination.

subnet

FreeBSD will automatically add subnet routes for the local subnet. In this example, `10.20.30.255` is the broadcast address for the subnet `10.20.30` and `example.com` is the domain name associated with that subnet. The designation `link#1` refers to the first Ethernet card in the machine.

Local network hosts and local subnets have their routes automatically configured by a daemon called routed(8). If it is not running, only routes which are statically defined by the administrator will exist.

host

The `host1` line refers to the host by its Ethernet address. Since it is the sending host, FreeBSD knows to use the loopback interface (lo0) rather than the Ethernet interface.

The two `host2` lines represent aliases which were created using ifconfig(8). The ⇒ symbol after the lo0 interface says that an alias has been set in addition to the loopback address. Such routes only show up on the host that supports the alias and all other hosts on the local network will have a `link#1` line for such routes.

224

The final line (destination subnet `224`) deals with multicasting.

Various attributes of each route can be seen in the `Flags` column. Commonly Seen Routing Table Flags summarizes some of these flags and their meanings:

Table 50. Commonly Seen Routing Table Flags

Flag	Purpose
U	The route is active (up).
H	The route destination is a single host.
G	Send anything for this destination on to this gateway, which will figure out from there where to send it.
S	This route was statically configured.
C	Clones a new route based upon this route for machines to connect to. This type of route is normally used for local networks.
W	The route was auto-configured based upon a local area network (clone) route.
L	Route involves references to Ethernet (link) hardware.

On a FreeBSD system, the default route can be defined in /etc/rc.conf by specifying the IP address of the default gateway:

```
defaultrouter="10.20.30.1"
```

It is also possible to manually add the route using `route`:

```
# route add default 10.20.30.1
```

Note that manually added routes will not survive a reboot. For more information on manual manipulation of network routing tables, refer to route(8).

35.2.2. Configuring a Router with Static Routes

A FreeBSD system can be configured as the default gateway, or router, for a network if it is a dual-homed system. A dual-homed system is a host which resides on at least two different networks. Typically, each network is connected to a separate network interface, though IP aliasing can be used to bind multiple addresses, each on a different subnet, to one physical interface.

In order for the system to forward packets between interfaces, FreeBSD must be configured as a router. Internet standards and good engineering practice prevent the FreeBSD Project from enabling this feature by default, but it can be configured to start at boot by adding this line to /etc/rc.conf:

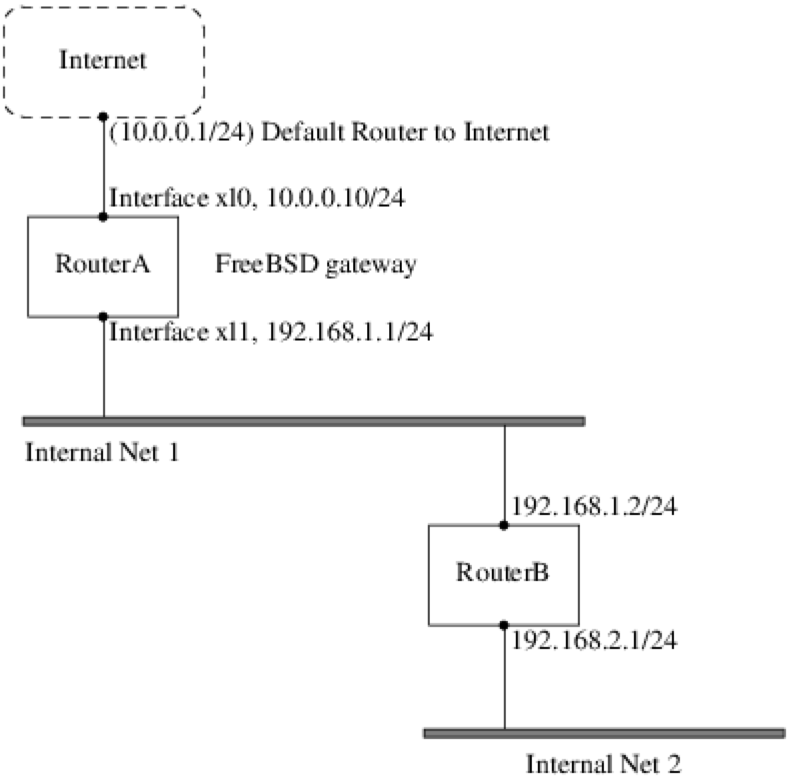
```
gateway_enable="YES"          # Set to YES if this host will be a gateway
```

To enable routing now, set the sysctl(8) variable `net.inet.ip.forwarding` to `1`. To stop routing, reset this variable to `0`.

The routing table of a router needs additional routes so it knows how to reach other networks. Routes can be either added manually using static routes or routes can be automatically learned using a routing protocol. Static routes are appropriate for small networks and this section describes how to add a static routing entry for a small network.

For large networks, static routes quickly become unscalable. FreeBSD comes with the standard BSD routing daemon routed(8), which provides the routing protocols RIP, versions 1 and 2, and IRDP. Support for the BGP and OSPF routing protocols can be installed using the net/quagga package or port.

Consider the following network:


Internet
(10.0.0.1/24) Default Router to Internet
Interface xl0, 10.0.0.10/24
RouterA
FreeBSD gateway
Interface xl1, 192.168.1.1/24
Internal Net 1
192.168.1.2/24
RouterB
192.168.2.1/24
Internal Net 2

In this scenario, `RouterA` is a FreeBSD machine that is acting as a router to the rest of the Internet. It has a default route set to `10.0.0.1` which allows it to connect with the outside world. `RouterB` is already configured to use `192.168.1.1` as its default gateway.

Before adding any static routes, the routing table on `RouterA` looks like this:

```
% netstat -nr
Routing tables

Internet:
Destination        Gateway            Flags    Refs      Use  Netif  Expire
default            10.0.0.1           UGS         0    49378    xl0
127.0.0.1          127.0.0.1          UH          0        6    lo0
10.0.0.0/24        link#1             UC          0        0    xl0
192.168.1.0/24     link#2             UC          0        0    xl1
```

With the current routing table, `RouterA` does not have a route to the `192.168.2.0/24` network. The following command adds the `Internal Net 2` network to `RouterA`'s routing table using `192.168.1.2` as the next hop:

```
# route add -net 192.168.2.0/24 192.168.1.2
```

Now, `RouterA` can reach any host on the `192.168.2.0/24` network. However, the routing information will not persist if the FreeBSD system reboots. If a static route needs to be persistent, add it to /etc/rc.conf:

```
# Add Internal Net 2 as a persistent static route
```

```
static_routes="internalnet2"
route_internalnet2="-net 192.168.2.0/24 192.168.1.2"
```

The `static_routes` configuration variable is a list of strings separated by a space, where each string references a route name. The variable `route_internalnet2` contains the static route for that route name.

Using more than one string in `static_routes` creates multiple static routes. The following shows an example of adding static routes for the `192.168.0.0/24` and `192.168.1.0/24` networks:

```
static_routes="net1 net2"
route_net1="-net 192.168.0.0/24 192.168.0.1"
route_net2="-net 192.168.1.0/24 192.168.1.1"
```

35.2.3. Troubleshooting

When an address space is assigned to a network, the service provider configures their routing tables so that all traffic for the network will be sent to the link for the site. But how do external sites know to send their packets to the network's ISP?

There is a system that keeps track of all assigned address spaces and defines their point of connection to the Internet backbone, or the main trunk lines that carry Internet traffic across the country and around the world. Each backbone machine has a copy of a master set of tables, which direct traffic for a particular network to a specific backbone carrier, and from there down the chain of service providers until it reaches a particular network.

It is the task of the service provider to advertise to the backbone sites that they are the point of connection, and thus the path inward, for a site. This is known as route propagation.

Sometimes, there is a problem with route propagation and some sites are unable to connect. Perhaps the most useful command for trying to figure out where routing is breaking down is `traceroute`. It is useful when `ping` fails.

When using `traceroute`, include the address of the remote host to connect to. The output will show the gateway hosts along the path of the attempt, eventually either reaching the target host, or terminating because of a lack of connection. For more information, refer to traceroute(8).

35.2.4. Multicast Considerations

FreeBSD natively supports both multicast applications and multicast routing. Multicast applications do not require any special configuration in order to run on FreeBSD. Support for multicast routing requires that the following option be compiled into a custom kernel:

```
options MROUTING
```

The multicast routing daemon, mrouted can be installed using the net/mrouted package or port. This daemon implements the DVMRP multicast routing protocol and is configured by editing

/usr/local/etc/mrouted.conf in order to set up the tunnels and DVMRP. The installation of mrouted also installs map-mbone and mrinfo, as well as their associated man pages. Refer to these for configuration examples.

> DVMRP has largely been replaced by the PIM protocol in many multicast installations. Refer to pim(4) for more information.

35.3. Virtual Hosts

A common use of FreeBSD is virtual site hosting, where one server appears to the network as many servers. This is achieved by assigning multiple network addresses to a single interface.

A given network interface has one "real" address, and may have any number of "alias" addresses. These aliases are normally added by placing alias entries in /etc/rc.conf, as seen in this example:

```
# sysrc ifconfig_fxp0_alias0="inet xxx.xxx.xxx.xxx netmask xxx.xxx.xxx.xxx"
```

Alias entries must start with `alias0` using a sequential number such as `alias0`, `alias1`, and so on. The configuration process will stop at the first missing number.

The calculation of alias netmasks is important. For a given interface, there must be one address which correctly represents the network's netmask. Any other addresses which fall within this network must have a netmask of all `1`s, expressed as either `255.255.255.255` or `0xffffffff`.

For example, consider the case where the `fxp0` interface is connected to two networks: `10.1.1.0` with a netmask of `255.255.255.0` and `202.0.75.16` with a netmask of `255.255.255.240`. The system is to be configured to appear in the ranges `10.1.1.1` through `10.1.1.5` and `202.0.75.17` through `202.0.75.20`. Only the first address in a given network range should have a real netmask. All the rest (`10.1.1.2` through `10.1.1.5` and `202.0.75.18` through `202.0.75.20`) must be configured with a netmask of `255.255.255.255`.

The following /etc/rc.conf entries configure the adapter correctly for this scenario:

```
# sysrc ifconfig_fxp0="inet 10.1.1.1 netmask 255.255.255.0"
# sysrc ifconfig_fxp0_alias0="inet 10.1.1.2 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias1="inet 10.1.1.3 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias2="inet 10.1.1.4 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias3="inet 10.1.1.5 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias4="inet 202.0.75.17 netmask 255.255.255.240"
# sysrc ifconfig_fxp0_alias5="inet 202.0.75.18 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias6="inet 202.0.75.19 netmask 255.255.255.255"
# sysrc ifconfig_fxp0_alias7="inet 202.0.75.20 netmask 255.255.255.255"
```

A simpler way to express this is with a space-separated list of IP address ranges. The first address will be given the indicated subnet mask and the additional addresses will have a subnet mask of `255.255.255.255`.

```
# sysrc ifconfig_fxp0_aliases="inet 10.1.1.1-5/24 inet 202.0.75.17-20/28"
```

35.4. Wireless Advanced Authentication

FreeBSD supports different ways of connecting to a wireless network. This section describes how to perform advanced authentication to a Wireless Network.

To make a connection and basic authentication to a wireless network the section Connection and Authentication to a Wireless Network in the Network Chapter describes how to do it.

35.4.1. WPA with EAP-TLS

The second way to use WPA is with an 802.1X backend authentication server. In this case, WPA is called WPA Enterprise to differentiate it from the less secure WPA Personal. Authentication in WPA Enterprise is based on the Extensible Authentication Protocol (EAP).

EAP does not come with an encryption method. Instead, EAP is embedded inside an encrypted tunnel. There are many EAP authentication methods, but EAP-TLS, EAP-TTLS, and EAP-PEAP are the most common.

EAP with Transport Layer Security (EAP-TLS) is a well-supported wireless authentication protocol since it was the first EAP method to be certified by the Wi-Fi Alliance. EAP-TLS requires three certificates to run: the certificate of the Certificate Authority (CA) installed on all machines, the server certificate for the authentication server, and one client certificate for each wireless client. In this EAP method, both the authentication server and wireless client authenticate each other by presenting their respective certificates, and then verify that these certificates were signed by the organization’s CA.

As previously, the configuration is done via /etc/wpa_supplicant.conf:

```
network={
  ssid="freebsdap" ①
  proto=RSN  ②
  key_mgmt=WPA-EAP ③
  eap=TLS ④
  identity="loader" ⑤
  ca_cert="/etc/certs/cacert.pem" ⑥
  client_cert="/etc/certs/clientcert.pem" ⑦
  private_key="/etc/certs/clientkey.pem" ⑧
  private_key_passwd="freebsdmallclient" ⑨
}
```

① This field indicates the network name (SSID).

② This example uses the RSN IEEE® 802.11i protocol, also known as WPA2.

③ The `key_mgmt` line refers to the key management protocol to use. In this example, it is WPA using EAP authentication.

④ This field indicates the EAP method for the connection.

⑤ The `identity` field contains the identity string for EAP.

⑥ The `ca_cert` field indicates the pathname of the CA certificate file. This file is needed to verify the server certificate.

⑦ The `client_cert` line gives the pathname to the client certificate file. This certificate is unique to each wireless client of the network.

⑧ The `private_key` field is the pathname to the client certificate private key file.

⑨ The `private_key_passwd` field contains the passphrase for the private key.

Then, add the following lines to /etc/rc.conf:

```
wlans_ath0="wlan0"
ifconfig_wlan0="WPA DHCP"
```

The next step is to bring up the interface:

```
# service netif start
Starting wpa_supplicant.
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 15
DHCPACK from 192.168.0.20
bound to 192.168.0.254 -- renewal in 300 seconds.
wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether 00:11:95:d5:43:62
      inet 192.168.0.254 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet DS/11Mbps mode 11g
      status: associated
      ssid freebsdap channel 1 (2412 Mhz 11g) bssid 00:11:95:c3:0d:ac
      country US ecm authmode WPA2/802.11i privacy ON deftxkey UNDEF
      AES-CCM 3:128-bit txpower 21.5 bmiss 7 scanvalid 450 bgscan
      bgscanintvl 300 bgscanidle 250 roam:rssi 7 roam:rate 5 protmode CTS
      wme burst roaming MANUAL
```

It is also possible to bring up the interface manually using wpa_supplicant(8) and ifconfig(8).

35.4.2. WPA with EAP-TTLS

With EAP-TLS, both the authentication server and the client need a certificate. With EAP-TTLS, a client certificate is optional. This method is similar to a web server which creates a secure SSL tunnel even if visitors do not have client-side certificates. EAP-TTLS uses an encrypted TLS tunnel for safe transport of the authentication data.

The required configuration can be added to /etc/wpa_supplicant.conf:

```
network={
  ssid="freebsdap"
  proto=RSN
  key_mgmt=WPA-EAP
  eap=TTLS ①
  identity="test" ②
  password="test" ③
  ca_cert="/etc/certs/cacert.pem" ④
  phase2="auth=MD5" ⑤
}
```

① This field specifies the EAP method for the connection.

② The `identity` field contains the identity string for EAP authentication inside the encrypted TLS tunnel.

③ The `password` field contains the passphrase for the EAP authentication.

④ The `ca_cert` field indicates the pathname of the CA certificate file. This file is needed to verify the server certificate.

⑤ This field specifies the authentication method used in the encrypted TLS tunnel. In this example, EAP with MD5-Challenge is used. The "inner authentication" phase is often called "phase2".

Next, add the following lines to /etc/rc.conf:

```
wlans_ath0="wlan0"
ifconfig_wlan0="WPA DHCP"
```

The next step is to bring up the interface:

```
# service netif start
Starting wpa_supplicant.
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 15
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 21
DHCPACK from 192.168.0.20
bound to 192.168.0.254 -- renewal in 300 seconds.
wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether 00:11:95:d5:43:62
      inet 192.168.0.254 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet DS/11Mbps mode 11g
      status: associated
      ssid freebsdap channel 1 (2412 Mhz 11g) bssid 00:11:95:c3:0d:ac
      country US ecm authmode WPA2/802.11i privacy ON deftxkey UNDEF
      AES-CCM 3:128-bit txpower 21.5 bmiss 7 scanvalid 450 bgscan
      bgscanintvl 300 bgscanidle 250 roam:rssi 7 roam:rate 5 protmode CTS
      wme burst roaming MANUAL
```

35.4.3. WPA with EAP-PEAP

PEAPv0/EAP-MSCHAPv2 is the most common PEAP method. In this chapter, the term PEAP is used to refer to that method.

Protected EAP (PEAP) is designed as an alternative to EAP-TTLS and is the most used EAP standard after EAP-TLS. In a network with mixed operating systems, PEAP should be the most supported standard after EAP-TLS.

PEAP is similar to EAP-TTLS as it uses a server-side certificate to authenticate clients by creating an encrypted TLS tunnel between the client and the authentication server, which protects the ensuing exchange of authentication information. PEAP authentication differs from EAP-TTLS as it broadcasts the username in the clear and only the password is sent in the encrypted TLS tunnel. EAP-TTLS will use the TLS tunnel for both the username and password.

Add the following lines to /etc/wpa_supplicant.conf to configure the EAP-PEAP related settings:

```
network={
  ssid="freebsdap"
  proto=RSN
  key_mgmt=WPA-EAP
  eap=PEAP ①
  identity="test" ②
  password="test" ③
  ca_cert="/etc/certs/cacert.pem" ④
  phase1="peaplabel=0" ⑤
  phase2="auth=MSCHAPV2" ⑥
}
```

① This field specifies the EAP method for the connection.

② The `identity` field contains the identity string for EAP authentication inside the encrypted TLS tunnel.

③ The `password` field contains the passphrase for the EAP authentication.

④ The `ca_cert` field indicates the pathname of the CA certificate file. This file is needed to verify the server certificate.

⑤ This field contains the parameters for the first phase of authentication, the TLS tunnel. According to the authentication server used, specify a specific label for authentication. Most of the time, the label will be "client EAP encryption" which is set by using `peaplabel=0`. More information can be found in wpa_supplicant.conf(5).

⑥ This field specifies the authentication protocol used in the encrypted TLS tunnel. In the case of PEAP, it is `auth=MSCHAPV2`.

Add the following to /etc/rc.conf:

```
wlans_ath0="wlan0"
ifconfig_wlan0="WPA DHCP"
```

Then, bring up the interface:

```
# service netif start
Starting wpa_supplicant.
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 15
DHCPREQUEST on wlan0 to 255.255.255.255 port 67 interval 21
DHCPACK from 192.168.0.20
bound to 192.168.0.254 -- renewal in 300 seconds.
wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      ether 00:11:95:d5:43:62
      inet 192.168.0.254 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet DS/11Mbps mode 11g
      status: associated
      ssid freebsdap channel 1 (2412 Mhz 11g) bssid 00:11:95:c3:0d:ac
      country US ecm authmode WPA2/802.11i privacy ON deftxkey UNDEF
      AES-CCM 3:128-bit txpower 21.5 bmiss 7 scanvalid 450 bgscan
      bgscanintvl 300 bgscanidle 250 roam:rssi 7 roam:rate 5 protmode CTS
      wme burst roaming MANUAL
```

35.5. Wireless Ad-hoc Mode

IBSS mode, also called ad-hoc mode, is designed for point to point connections. For example, to establish an ad-hoc network between the machines `A` and `B`, choose two IP addresses and a SSID.

On `A`:

```
# ifconfig wlan0 create wlandev ath0 wlanmode adhoc
# ifconfig wlan0 inet 192.168.0.1 netmask 255.255.255.0 ssid freebsdap
# ifconfig wlan0
  wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
      ether 00:11:95:c3:0d:ac
      inet 192.168.0.1 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet autoselect mode 11g <adhoc>
      status: running
      ssid freebsdap channel 2 (2417 Mhz 11g) bssid 02:11:95:c3:0d:ac
      country US ecm authmode OPEN privacy OFF txpower 21.5 scanvalid 60
      protmode CTS wme burst
```

The `adhoc` parameter indicates that the interface is running in IBSS mode.

`B` should now be able to detect `A`:

```
# ifconfig wlan0 create wlandev ath0 wlanmode adhoc
# ifconfig wlan0 up scan
  SSID/MESH ID    BSSID              CHAN RATE    S:N     INT CAPS
  freebsdap       02:11:95:c3:0d:ac    2   54M -64:-96   100 IS   WME
```

The `I` in the output confirms that `A` is in ad-hoc mode. Now, configure `B` with a different IP address:

```
# ifconfig wlan0 inet 192.168.0.2 netmask 255.255.255.0 ssid freebsdap
# ifconfig wlan0
  wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
      ether 00:11:95:d5:43:62
      inet 192.168.0.2 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet autoselect mode 11g <adhoc>
      status: running
      ssid freebsdap channel 2 (2417 Mhz 11g) bssid 02:11:95:c3:0d:ac
      country US ecm authmode OPEN privacy OFF txpower 21.5 scanvalid 60
      protmode CTS wme burst
```

Both `A` and `B` are now ready to exchange information.

35.5.1. FreeBSD Host Access Points

FreeBSD can act as an Access Point (AP) which eliminates the need to buy a hardware AP or run an ad-hoc network. This can be particularly useful when a FreeBSD machine is acting as a gateway to another network such as the Internet.

Basic Settings

Before configuring a FreeBSD machine as an AP, the kernel must be configured with the appropriate networking support for the wireless card as well as the security protocols being used. For more details, see Basic Settings.

ℹ The NDIS driver wrapper for Windows® drivers does not currently support AP operation. Only native FreeBSD wireless drivers support AP mode.

Once wireless networking support is loaded, check if the wireless device supports the host-based access point mode, also known as hostap mode:

```
# ifconfig wlan0 create wlandev ath0
# ifconfig wlan0 list caps
drivercaps=6f85edc1<STA,FF,TURBOP,IBSS,HOSTAP,AHDEMO,TXPMGT,SHSLOT,SHPREAMBLE,MONITOR,
MBSS,WPA1,WPA2,BURST,WME,WDS,BGSCAN,TXFRAG>
cryptocaps=1f<WEP,TKIP,AES,AES_CCM,TKIPMIC>
```

This output displays the card's capabilities. The `HOSTAP` word confirms that this wireless card can act as an AP. Various supported ciphers are also listed: WEP, TKIP, and AES. This information indicates which security protocols can be used on the AP.

The wireless device can only be put into hostap mode during the creation of the network pseudo-device, so a previously created device must be destroyed first:

```
# ifconfig wlan0 destroy
```

then regenerated with the correct option before setting the other parameters:

```
# ifconfig wlan0 create wlandev ath0 wlanmode hostap
# ifconfig wlan0 inet 192.168.0.1 netmask 255.255.255.0 ssid freebsdap mode 11g
channel 1
```

Use ifconfig(8) again to see the status of the wlan0 interface:

```
# ifconfig wlan0
  wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
      ether 00:11:95:c3:0d:ac
      inet 192.168.0.1 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet autoselect mode 11g <hostap>
      status: running
      ssid freebsdap channel 1 (2412 Mhz 11g) bssid 00:11:95:c3:0d:ac
      country US ecm authmode OPEN privacy OFF txpower 21.5 scanvalid 60
      protmode CTS wme burst dtimperiod 1 -dfs
```

The `hostap` parameter indicates the interface is running in the host-based access point mode.

The interface configuration can be done automatically at boot time by adding the following lines to /etc/rc.conf:

```
wlans_ath0="wlan0"
create_args_wlan0="wlanmode hostap"
ifconfig_wlan0="inet 192.168.0.1 netmask 255.255.255.0 ssid freebsdap mode 11g channel
1"
```

Host-based Access Point Without Authentication or Encryption

Although it is not recommended to run an AP without any authentication or encryption, this is a simple way to check if the AP is working. This configuration is also important for debugging client issues.

Once the AP is configured, initiate a scan from another wireless machine to find the AP:

```
# ifconfig wlan0 create wlandev ath0
# ifconfig wlan0 up scan
SSID/MESH ID    BSSID              CHAN RATE   S:N     INT CAPS
freebsdap       00:11:95:c3:0d:ac    1   54M -66:-96  100 ES   WME
```

The client machine found the AP and can be associated with it:

```
# ifconfig wlan0 inet 192.168.0.2 netmask 255.255.255.0 ssid freebsdap
# ifconfig wlan0
  wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
      ether 00:11:95:d5:43:62
      inet 192.168.0.2 netmask 0xffffff00 broadcast 192.168.0.255
      media: IEEE 802.11 Wireless Ethernet OFDM/54Mbps mode 11g
      status: associated
      ssid freebsdap channel 1 (2412 Mhz 11g) bssid 00:11:95:c3:0d:ac
      country US ecm authmode OPEN privacy OFF txpower 21.5 bmiss 7
      scanvalid 60 bgscan bgscanintvl 300 bgscanidle 250 roam:rssi 7
      roam:rate 5 protmode CTS wme burst
```

WPA2 Host-based Access Point

This section focuses on setting up a FreeBSD access point using the WPA2 security protocol. More details regarding WPA and the configuration of WPA-based wireless clients can be found in WPA with EAP-TLS.

The hostapd(8) daemon is used to deal with client authentication and key management on the WPA2-enabled AP.

The following configuration operations are performed on the FreeBSD machine acting as the AP. Once the AP is correctly working, hostapd(8) can be automatically started at boot with this line in /etc/rc.conf:

```
hostapd_enable="YES"
```

Before trying to configure hostapd(8), first configure the basic settings introduced in Basic Settings.

WPA2-PSK

WPA2-PSK is intended for small networks where the use of a backend authentication server is not possible or desired.

The configuration is done in /etc/hostapd.conf:

```
interface=wlan0                    ①
debug=1                            ②
ctrl_interface=/var/run/hostapd    ③
ctrl_interface_group=wheel         ④
ssid=freebsdap                     ⑤
wpa=2                              ⑥
wpa_passphrase=freebsdmall         ⑦
wpa_key_mgmt=WPA-PSK               ⑧
wpa_pairwise=CCMP                  ⑨
```

① Wireless interface used for the access point.

② Level of verbosity used during the execution of hostapd(8). A value of `1` represents the minimal level.

③ Pathname of the directory used by hostapd(8) to store domain socket files for communication with external programs such as hostapd_cli(8). The default value is used in this example.

④ The group allowed to access the control interface files.

⑤ The wireless network name, or SSID, that will appear in wireless scans.

⑥ Enable WPA and specify which WPA authentication protocol will be required. A value of `2` configures the AP for WPA2 and is recommended. Set to `1` only if the obsolete WPA is required.

⑦ ASCII passphrase for WPA authentication.

⑧ The key management protocol to use. This example sets WPA-PSK.

⑨ Encryption algorithms accepted by the access point. In this example, only the CCMP (AES) cipher is accepted. CCMP is an alternative to TKIP and is strongly preferred when possible. TKIP should be allowed only when there are stations incapable of using CCMP.

The next step is to start hostapd(8):

```
# service hostapd forcestart
```

```
# ifconfig wlan0
wlan0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> metric 0 mtu 1500
    ether 04:f0:21:16:8e:10
    inet6 fe80::6f0:21ff:fe16:8e10%wlan0 prefixlen 64 scopeid 0x9
    nd6 options=21<PERFORMNUD,AUTO_LINKLOCAL>
    media: IEEE 802.11 Wireless Ethernet autoselect mode 11na <hostap>
    status: running
    ssid No5ignal channel 36 (5180 MHz 11a ht/40+) bssid 04:f0:21:16:8e:10
    country US ecm authmode WPA2/802.11i privacy MIXED deftxkey 2
    AES-CCM 2:128-bit AES-CCM 3:128-bit txpower 17 mcastrate 6 mgmtrate 6
    scanvalid 60 ampdulimit 64k ampdudensity 8 shortgi wme burst
    dtimperiod 1 -dfs
    groups: wlan
```

Once the AP is running, the clients can associate with it. See Basic Settings for more details. It is possible to see the stations associated with the AP using `ifconfig wlan0 list sta`.

35.6. USB Tethering

Many cellphones provide the option to share their data connection over USB (often called "tethering"). This feature uses one of RNDIS, CDC, or a custom Apple® iPhone®/iPad® protocol.

- Android™ devices generally use the urndis(4) driver.
- Apple® devices use the ipheth(4) driver.
- Older devices will often use the cdce(4) driver.

Before attaching a device, load the appropriate driver into the kernel:

```
# kldload if_urndis
# kldload if_cdce
# kldload if_ipheth
```

Once the device is attached `ue`*0* will be available for use like a normal network device. Be sure that the "USB tethering" option is enabled on the device.

To make this change permanent and load the driver as a module at boot time, place the appropriate line of the following in /boot/loader.conf:

```
if_urndis_load="YES"
if_cdce_load="YES"
if_ipheth_load="YES"
```

35.7. Bluetooth

Bluetooth is a wireless technology for creating personal networks operating in the 2.4 GHz unlicensed band, with a range of 10 meters. Networks are usually formed ad-hoc from portable devices such as cellular phones, handhelds, and laptops. Unlike Wi-Fi wireless technology, Bluetooth offers higher level service profiles, such as FTP-like file servers, file pushing, voice transport, serial line emulation, and more.

This section describes the use of a USB Bluetooth dongle on a FreeBSD system. It then describes the various Bluetooth protocols and utilities.

35.7.1. Loading Bluetooth Support

The Bluetooth stack in FreeBSD is implemented using the netgraph(4) framework. A broad variety of Bluetooth USB dongles is supported by ng_ubt(4). Broadcom BCM2033 based Bluetooth devices are supported by the ubtbcmfw(4) and ng_ubt(4) drivers. The 3Com Bluetooth PC Card 3CRWB60-A is supported by the ng_bt3c(4) driver. Serial and UART based Bluetooth devices are supported by ng_h4(4) and hcseriald(8).

Before attaching a device, determine which of the above drivers it uses, then load the driver. For example, if the device uses the ng_ubt(4) driver:

```
# kldload ng_ubt
```

If the Bluetooth device will be attached to the system during system startup, the system can be configured to load the module at boot time by adding the driver to /boot/loader.conf:

```
ng_ubt_load="YES"
```

Once the driver is loaded, plug in the USB dongle. If the driver load was successful, output similar to the following should appear on the console and in /var/log/messages:

```
ubt0: vendor 0x0a12 product 0x0001, rev 1.10/5.25, addr 2
ubt0: Interface 0 endpoints: interrupt=0x81, bulk-in=0x82, bulk-out=0x2
ubt0: Interface 1 (alt.config 5) endpoints: isoc-in=0x83, isoc-out=0x3,
      wMaxPacketSize=49, nframes=6, buffer size=294
```

To start and stop the Bluetooth stack, use its startup script. It is a good idea to stop the stack before unplugging the device. Starting the bluetooth stack might require hcsecd(8) to be started. When starting the stack, the output should be similar to the following:

```
# service bluetooth start ubt0
BD_ADDR: 00:02:72:00:d4:1a
Features: 0xff 0xff 0xf 00 00 00 00 00
<3-Slot> <5-Slot> <Encryption> <Slot offset>
<Timing accuracy> <Switch> <Hold mode> <Sniff mode>
<Park mode> <RSSI> <Channel quality> <SCO link>
<HV2 packets> <HV3 packets> <u-law log> <A-law log> <CVSD>
<Paging scheme> <Power control> <Transparent SCO data>
Max. ACL packet size: 192 bytes
Number of ACL packets: 8
Max. SCO packet size: 64 bytes
Number of SCO packets: 8
```

35.7.2. Finding Other Bluetooth Devices

The Host Controller Interface (HCI) provides a uniform method for accessing Bluetooth baseband capabilities. In FreeBSD, a netgraph HCI node is created for each Bluetooth device. For more details, refer to ng_hci(4).

One of the most common tasks is discovery of Bluetooth devices within RF proximity. This operation is called *inquiry*. Inquiry and other HCI related operations are done using hccontrol(8). The example below shows how to find out which Bluetooth devices are in range. The list of devices should be displayed in a few seconds. Note that a remote device will only answer the inquiry if it is set to *discoverable* mode.

```
% hccontrol -n ubt0hci inquiry
Inquiry result, num_responses=1
Inquiry result #0
       BD_ADDR: 00:80:37:29:19:a4
       Page Scan Rep. Mode: 0x1
       Page Scan Period Mode: 00
       Page Scan Mode: 00
       Class: 52:02:04
       Clock offset: 0x78ef

```

```
Inquiry complete. Status: No error [00]
```

The `BD_ADDR` is the unique address of a Bluetooth device, similar to the MAC address of a network card. This address is needed for further communication with a device and it is possible to assign a human readable name to a `BD_ADDR`. Information regarding the known Bluetooth hosts is contained in /etc/bluetooth/hosts. The following example shows how to obtain the human readable name that was assigned to the remote device:

```
% hccontrol -n ubt0hci remote_name_request 00:80:37:29:19:a4
BD_ADDR: 00:80:37:29:19:a4
Name: Pav's T39
```

If an inquiry is performed on a remote Bluetooth device, it will find the computer as "your.host.name (ubt0)". The name assigned to the local device can be changed at any time.

Remote devices can be assigned aliases in /etc/bluetooth/hosts. More information about /etc/bluetooth/hosts file might be found in bluetooth.hosts(5).

The Bluetooth system provides a point-to-point connection between two Bluetooth units, or a point-to-multipoint connection which is shared among several Bluetooth devices. The following example shows how to create a connection to a remote device:

```
% hccontrol -n ubt0hci create_connection BT_ADDR
```

`create_connection` accepts `BT_ADDR` as well as host aliases in /etc/bluetooth/hosts.

The following example shows how to obtain the list of active baseband connections for the local device:

```
% hccontrol -n ubt0hci read_connection_list
Remote BD_ADDR    Handle Type Mode Role Encrypt Pending Queue State
00:80:37:29:19:a4     41  ACL    0 MAST    NONE       0     0 OPEN
```

A *connection handle* is useful when termination of the baseband connection is required, though it is normally not required to do this by hand. The stack will automatically terminate inactive baseband connections.

```
# hccontrol -n ubt0hci disconnect 41
Connection handle: 41
Reason: Connection terminated by local host [0x16]
```

Type `hccontrol help` for a complete listing of available HCI commands. Most of the HCI commands do not require superuser privileges.

35.7.3. Device Pairing

By default, Bluetooth communication is not authenticated, and any device can talk to any other device. A Bluetooth device, such as a cellular phone, may choose to require authentication to provide a particular service. Bluetooth authentication is normally done with a *PIN code*, an ASCII string up to 16 characters in length. The user is required to enter the same PIN code on both devices. Once the user has entered the PIN code, both devices will generate a *link key*. After that, the link key can be stored either in the devices or in a persistent storage. Next time, both devices will use the previously generated link key. This procedure is called *pairing*. Note that if the link key is lost by either device, the pairing must be repeated.

The hcsecd(8) daemon is responsible for handling Bluetooth authentication requests. The default configuration file is /etc/bluetooth/hcsecd.conf. An example section for a cellular phone with the PIN code set to `1234` is shown below:

```
device {
        bdaddr  00:80:37:29:19:a4;
        name    "Pav's T39";
        key     nokey;
        pin     "1234";
      }
```

The only limitation on PIN codes is length. Some devices, such as Bluetooth headsets, may have a fixed PIN code built in. The `-d` switch forces hcsecd(8) to stay in the foreground, so it is easy to see what is happening. Set the remote device to receive pairing and initiate the Bluetooth connection to the remote device. The remote device should indicate that pairing was accepted and request the PIN code. Enter the same PIN code listed in hcsecd.conf. Now the computer and the remote device are paired. Alternatively, pairing can be initiated on the remote device.

The following line can be added to /etc/rc.conf to configure hcsecd(8) to start automatically on system start:

```
hcsecd_enable="YES"
```

The following is a sample of the hcsecd(8) daemon output:

```
hcsecd[16484]: Got Link_Key_Request event from 'ubt0hci', remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Found matching entry, remote bdaddr 0:80:37:29:19:a4, name 'Pav's T39',
link key doesn't exist
hcsecd[16484]: Sending Link_Key_Negative_Reply to 'ubt0hci' for remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Got PIN_Code_Request event from 'ubt0hci', remote bdaddr
0:80:37:29:19:a4
hcsecd[16484]: Found matching entry, remote bdaddr 0:80:37:29:19:a4, name 'Pav's T39',
PIN code exists
```

```
hcsecd[16484]: Sending PIN_Code_Reply to 'ubt0hci' for remote bdaddr 0:80:37:29:19:a4
```

35.7.4. Network Access with PPP Profiles

A Dial-Up Networking (DUN) profile can be used to configure a cellular phone as a wireless modem for connecting to a dial-up Internet access server. It can also be used to configure a computer to receive data calls from a cellular phone.

Network access with a PPP profile can be used to provide LAN access for a single Bluetooth device or multiple Bluetooth devices. It can also provide PC to PC connection using PPP networking over serial cable emulation.

In FreeBSD, these profiles are implemented with ppp(8) and the rfcomm_pppd(8) wrapper which converts a Bluetooth connection into something PPP can use. Before a profile can be used, a new PPP label must be created in /etc/ppp/ppp.conf. Consult rfcomm_pppd(8) for examples.

In this example, rfcomm_pppd(8) is used to open a connection to a remote device with a `BD_ADDR` of `00:80:37:29:19:a4` on a DUNRFCOMM channel:

```
# rfcomm_pppd -a 00:80:37:29:19:a4 -c -C dun -l rfcomm-dialup
```

The actual channel number will be obtained from the remote device using the SDP protocol. It is possible to specify the RFCOMM channel by hand, and in this case rfcomm_pppd(8) will not perform the SDP query. Use sdpcontrol(8) to find out the RFCOMM channel on the remote device.

In order to provide network access with the PPPLAN service, sdpd(8) must be running and a new entry for LAN clients must be created in /etc/ppp/ppp.conf. Consult rfcomm_pppd(8) for examples. Finally, start the RFCOMMPPP server on a valid RFCOMM channel number. The RFCOMMPPP server will automatically register the Bluetooth LAN service with the local SDP daemon. The example below shows how to start the RFCOMMPPP server.

```
# rfcomm_pppd -s -C 7 -l rfcomm-server
```

35.7.5. Bluetooth Protocols

This section provides an overview of the various Bluetooth protocols, their function, and associated utilities.

Logical Link Control and Adaptation Protocol (L2CAP)

The Logical Link Control and Adaptation Protocol (L2CAP) provides connection-oriented and connectionless data services to upper layer protocols. L2CAP permits higher level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

L2CAP is based around the concept of *channels*. A channel is a logical connection on top of a baseband connection, where each channel is bound to a single protocol in a many-to-one fashion. Multiple channels can be bound to the same protocol, but a channel cannot be bound to multiple protocols. Each L2CAP packet received on a channel is directed to the appropriate higher level protocol. Multiple channels can share the same baseband connection.

In FreeBSD, a netgraph L2CAP node is created for each Bluetooth device. This node is normally connected to the downstream Bluetooth HCI node and upstream Bluetooth socket nodes. The default name for the L2CAP node is "devicel2cap". For more details refer to ng_l2cap(4).

A useful command is l2ping(8), which can be used to ping other devices. Some Bluetooth implementations might not return all of the data sent to them, so `0 bytes` in the following example is normal.

```
# l2ping -a 00:80:37:29:19:a4
0 bytes from 0:80:37:29:19:a4 seq_no=0 time=48.633 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=1 time=37.551 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=2 time=28.324 ms result=0
0 bytes from 0:80:37:29:19:a4 seq_no=3 time=46.150 ms result=0
```

The l2control(8) utility is used to perform various operations on L2CAP nodes. This example shows how to obtain the list of logical connections (channels) and the list of baseband connections for the local device:

```
% l2control -a 00:02:72:00:d4:1a read_channel_list
L2CAP channels:
Remote BD_ADDR     SCID/ DCID   PSM  IMTU/ OMTU State
00:07:e0:00:0b:ca    66/   64     3   132/  672 OPEN
% l2control -a 00:02:72:00:d4:1a read_connection_list
L2CAP connections:
Remote BD_ADDR    Handle Flags Pending State
00:07:e0:00:0b:ca     41 O           0 OPEN
```

Another diagnostic tool is btsockstat(1). It is similar to netstat(1), but for Bluetooth network-related data structures. The example below shows the same logical connection as l2control(8) above.

```
% btsockstat
Active L2CAP sockets
PCB      Recv-Q Send-Q Local address/PSM       Foreign address   CID   State
c2afe900      0      0 00:02:72:00:d4:1a/3     00:07:e0:00:0b:ca 66    OPEN
Active RFCOMM sessions
L2PCB    PCB      Flag MTU   Out-Q DLCs State
c2afe900 c2b53380 1    127   0     Yes  OPEN
Active RFCOMM sockets
PCB      Recv-Q Send-Q Local address     Foreign address   Chan DLCI State
c2e8bc80      0    250 00:02:72:00:d4:1a 00:07:e0:00:0b:ca 3    6    OPEN
```

Radio Frequency Communication (RFCOMM)

The RFCOMM protocol provides emulation of serial ports over the L2CAP protocol. RFCOMM is a simple transport protocol, with additional provisions for emulating the 9 circuits of RS-232 (EIATIA-232-E) serial ports. It supports up to 60 simultaneous connections (RFCOMM channels) between two Bluetooth devices.

For the purposes of RFCOMM, a complete communication path involves two applications running on the communication endpoints with a communication segment between them. RFCOMM is intended to cover applications that make use of the serial ports of the devices in which they reside. The communication segment is a direct connect Bluetooth link from one device to another.

RFCOMM is only concerned with the connection between the devices in the direct connect case, or between the device and a modem in the network case. RFCOMM can support other configurations, such as modules that communicate via Bluetooth wireless technology on one side and provide a wired interface on the other side.

In FreeBSD, RFCOMM is implemented at the Bluetooth sockets layer.

Service Discovery Protocol (SDP)

The Service Discovery Protocol (SDP) provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

SDP involves communication between a SDP server and a SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing a SDP request. If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes, but it does not provide a mechanism for utilizing those services.

Normally, a SDP client searches for services based on some desired characteristics of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any prior information about the services. This process of looking for any offered services is called *browsing*.

The Bluetooth SDP server, sdpd(8), and command line client, sdpcontrol(8), are included in the standard FreeBSD installation. The following example shows how to perform a SDP browse query.

```
% sdpcontrol -a 00:01:03:fc:6e:ec browse
Record Handle: 00000000
Service Class ID List:
        Service Discovery Server (0x1000)
Protocol Descriptor List:
        L2CAP (0x0100)
                Protocol specific parameter #1: u/int/uuid16 1
                Protocol specific parameter #2: u/int/uuid16 1

Record Handle: 0x00000001
Service Class ID List:
        Browse Group Descriptor (0x1001)

Record Handle: 0x00000002
Service Class ID List:
        LAN Access Using PPP (0x1102)
Protocol Descriptor List:
        L2CAP (0x0100)
        RFCOMM (0x0003)
                Protocol specific parameter #1: u/int8/bool 1
Bluetooth Profile Descriptor List:
        LAN Access Using PPP (0x1102) ver. 1.0
```

Note that each service has a list of attributes, such as the RFCOMM channel. Depending on the service, the user might need to make note of some of the attributes. Some Bluetooth implementations do not support service browsing and may return an empty list. In this case, it is possible to search for the specific service. The example below shows how to search for the OBEX Object Push (OPUSH) service:

```
% sdpcontrol -a 00:01:03:fc:6e:ec search OPUSH
```

Offering services on FreeBSD to Bluetooth clients is done with the sdpd(8) server. The following line can be added to /etc/rc.conf:

```
sdpd_enable="YES"
```

Then the sdpd(8) daemon can be started with:

```
# service sdpd start
```

The local server application that wants to provide a Bluetooth service to remote clients will register the service with the local SDP daemon. An example of such an application is rfcomm_pppd(8). Once started, it will register the Bluetooth LAN service with the local SDP daemon.

The list of services registered with the local SDP server can be obtained by issuing a SDP browse query via the local control channel:

```
# sdpcontrol -l browse
```

OBEX Object Push (OPUSH)

Object Exchange (OBEX) is a widely used protocol for simple file transfers between mobile devices. Its main use is in infrared communication, where it is used for generic file transfers between notebooks or PDAs, and for sending business cards or calendar entries between cellular phones and other devices with Personal Information Manager (PIM) applications.

The OBEX server and client are implemented by obexapp, which can be installed using the comms/obexapp package or port.

The OBEX client is used to push and/or pull objects from the OBEX server. An example object is a business card or an appointment. The OBEX client can obtain the RFCOMM channel number from the remote device via SDP. This can be done by specifying the service name instead of the RFCOMM channel number. Supported service names are: `IrMC`, `FTRN`, and `OPUSH`. It is also possible to specify the RFCOMM channel as a number. Below is an example of an OBEX session where the device information object is pulled from the cellular phone, and a new object, the business card, is pushed into the phone's directory.

```
% obexapp -a 00:80:37:29:19:a4 -C IrMC
obex> get telecom/devinfo.txt devinfo-t39.txt
Success, response: OK, Success (0x20)
obex> put new.vcf
Success, response: OK, Success (0x20)
obex> di
Success, response: OK, Success (0x20)
```

In order to provide the OPUSH service, sdpd(8) must be running and a root folder, where all incoming objects will be stored, must be created. The default path to the root folder is /var/spool/obex. Finally, start the OBEX server on a valid RFCOMM channel number. The OBEX server will automatically register the OPUSH service with the local SDP daemon. The example below shows how to start the OBEX server.

```
# obexapp -s -C 10
```

Serial Port Profile (SPP)

The Serial Port Profile (SPP) allows Bluetooth devices to perform serial cable emulation. This profile allows legacy applications to use Bluetooth as a cable replacement, through a virtual serial port abstraction.

In FreeBSD, rfcomm_sppd(1) implements SPP and a pseudo tty is used as a virtual serial port abstraction. The example below shows how to connect to a remote device's serial port service. A RFCOMM channel does not have to be specified as rfcomm_sppd(1) can obtain it from the remote device via SDP. To override this, specify a RFCOMM channel on the command line.

```
# rfcomm_sppd -a 00:07:E0:00:0B:CA -t
rfcomm_sppd[94692]: Starting on /dev/pts/6...
/dev/pts/6
```

Once connected, the pseudo tty can be used as serial port:

```
# cu -l /dev/pts/6
```

The pseudo tty is printed on stdout and can be read by wrapper scripts:

```
PTS=`rfcomm_sppd -a 00:07:E0:00:0B:CA -t`
cu -l $PTS
```

35.7.6. Troubleshooting

By default, when FreeBSD is accepting a new connection, it tries to perform a role switch and become master. Some older Bluetooth devices which do not support role switching will not be able to connect. Since role switching is performed when a new connection is being established, it is not possible to ask the remote device if it supports role switching. However, there is a HCI option to disable role switching on the local side:

```
# hccontrol -n ubt0hci write_node_role_switch 0
```

To display Bluetooth packets, use the third-party package hcidump, which can be installed using the comms/hcidump package or port. This utility is similar to tcpdump(1) and can be used to display the contents of Bluetooth packets on the terminal and to dump the Bluetooth packets to a file.

35.8. Bridging

It is sometimes useful to divide a network, such as an Ethernet segment, into network segments without having to create IP subnets and use a router to connect the segments together. A device that connects two networks together in this fashion is called a "bridge".

A bridge works by learning the MAC addresses of the devices on each of its network interfaces. It forwards traffic between networks only when the source and destination MAC addresses are on different networks. In many respects, a bridge is like an Ethernet switch with very few ports. A FreeBSD system with multiple network interfaces can be configured to act as a bridge.

Bridging can be useful in the following situations:

Connecting Networks

The basic operation of a bridge is to join two or more network segments. There are many reasons to use a host-based bridge instead of networking equipment, such as cabling constraints or firewalling. A bridge can also connect a wireless interface running in hostap mode to a wired network and act as an access point.

Filtering/Traffic Shaping Firewall

A bridge can be used when firewall functionality is needed without routing or Network Address Translation (NAT).

An example is a small company that is connected via DSL or ISDN to an ISP. There are thirteen public IP addresses from the ISP and ten computers on the network. In this situation, using a router-based firewall is difficult because of subnetting issues. A bridge-based firewall can be configured without any IP addressing issues.

Network Tap

A bridge can join two network segments in order to inspect all Ethernet frames that pass between them using bpf(4) and tcpdump(1) on the bridge interface, or by sending a copy of all frames out on an additional interface known as a span port.

Layer 2 VPN

Two Ethernet networks can be joined across an IP link by bridging the networks to an EtherIP tunnel or a tap(4) based solution such as OpenVPN.

Layer 2 Redundancy

A network can be connected together with multiple links and use the Spanning Tree Protocol (STP) to block redundant paths.

This section describes how to configure a FreeBSD system as a bridge using if_bridge(4). A netgraph bridging driver is also available, and is described in ng_bridge(4).

Packet filtering can be used with any firewall package that hooks into the pfil(9) framework. The bridge can be used as a traffic shaper with altq(4) or dummynet(4).

35.8.1. Enabling the Bridge

In FreeBSD, if_bridge(4) is a kernel module which is automatically loaded by ifconfig(8) when creating a bridge interface. It is also possible to compile bridge support into a custom kernel by adding `device if_bridge` to the custom kernel configuration file.

The bridge is created using interface cloning. To create the bridge interface:

```
# ifconfig bridge create
bridge0
# ifconfig bridge0
bridge0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether 96:3d:4b:f1:79:7a
        id 00:00:00:00:00:00 priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 100 timeout 1200
        root id 00:00:00:00:00:00 priority 0 ifcost 0 port 0
```

When a bridge interface is created, it is automatically assigned a randomly generated Ethernet address. The `maxaddr` and `timeout` parameters control how many MAC addresses the bridge will keep in its forwarding table and how many seconds before each entry is removed after it is last seen. The other parameters control how STP operates.

Next, specify which network interfaces to add as members of the bridge. For the bridge to forward packets, all member interfaces and the bridge need to be up:

```
# ifconfig bridge0 addm fxp0 addm fxp1 up
# ifconfig fxp0 up
# ifconfig fxp1 up
```

The bridge can now forward Ethernet frames between fxp0 and fxp1. Add the following lines to /etc/rc.conf so the bridge is created at startup:

```
cloned_interfaces="bridge0"
ifconfig_bridge0="addm fxp0 addm fxp1 up"
ifconfig_fxp0="up"
ifconfig_fxp1="up"
```

If the bridge host needs an IP address, set it on the bridge interface, not on the member interfaces. The address can be set statically or via DHCP. This example sets a static IP address:

```
# ifconfig bridge0 inet 192.168.0.1/24
```

It is also possible to assign an IPv6 address to a bridge interface.

To make the changes permanent, add the addressing information to /etc/rc.conf.

This example uses a dynamic IPv6 address:

```
ifconfig_bridge0_ipv6="inet6 auto_linklocal accept_rtadv"
```

> When packet filtering is enabled, bridged packets will pass through the filter inbound on the originating interface on the bridge interface, and outbound on the appropriate interfaces. Either stage can be disabled. When direction of the packet flow is important, it is best to firewall on the member interfaces rather than the bridge itself.
>
> The bridge has several configurable settings for passing non-IP and IP packets, and layer2 firewalling with ipfw(8). See if_bridge(4) for more information.

35.8.2. Enabling Spanning Tree

For an Ethernet network to function properly, only one active path can exist between two devices. The STP protocol detects loops and puts redundant links into a blocked state. Should one of the active links fail, STP calculates a different tree and enables one of the blocked paths to restore connectivity to all points in the network.

The Rapid Spanning Tree Protocol (RSTP or 802.1w) provides backwards compatibility with legacy STP. RSTP provides faster convergence and exchanges information with neighboring switches to quickly transition to forwarding mode without creating loops. FreeBSD supports RSTP and STP as operating modes, with RSTP being the default mode.

STP can be enabled on member interfaces using ifconfig(8). For a bridge with fxp0 and fxp1 as the current interfaces, enable STP with:

```
# ifconfig bridge0 stp fxp0 stp fxp1
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether d6:cf:d5:a0:94:6d
        id 00:01:02:4b:d4:50 priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 100 timeout 1200
        root id 00:01:02:4b:d4:50 priority 32768 ifcost 0 port 0
        member: fxp0 flags=1c7<LEARNING,DISCOVER,STP,AUTOEDGE,PTP,AUTOPTP>
                port 3 priority 128 path cost 200000 proto rstp
                role designated state forwarding
        member: fxp1 flags=1c7<LEARNING,DISCOVER,STP,AUTOEDGE,PTP,AUTOPTP>
                port 4 priority 128 path cost 200000 proto rstp
                role designated state forwarding
```

This bridge has a spanning tree ID of `00:01:02:4b:d4:50` and a priority of `32768`. As the `root id` is the same, it indicates that this is the root bridge for the tree.

Another bridge on the network also has STP enabled:

```
bridge0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        ether 96:3d:4b:f1:79:7a
        id 00:13:d4:9a:06:7a priority 32768 hellotime 2 fwddelay 15
        maxage 20 holdcnt 6 proto rstp maxaddr 100 timeout 1200
        root id 00:01:02:4b:d4:50 priority 32768 ifcost 400000 port 4
        member: fxp0 flags=1c7<LEARNING,DISCOVER,STP,AUTOEDGE,PTP,AUTOPTP>
                port 4 priority 128 path cost 200000 proto rstp
                role root state forwarding
        member: fxp1 flags=1c7<LEARNING,DISCOVER,STP,AUTOEDGE,PTP,AUTOPTP>
                port 5 priority 128 path cost 200000 proto rstp
                role designated state forwarding
```

The line `root id 00:01:02:4b:d4:50 priority 32768 ifcost 400000 port 4` shows that the root bridge is `00:01:02:4b:d4:50` and has a path cost of `400000` from this bridge. The path to the root bridge is via `port 4` which is fxp0.

35.8.3. Bridge Interface Parameters

Several `ifconfig` parameters are unique to bridge interfaces. This section summarizes some common uses for these parameters. The complete list of available parameters is described in ifconfig(8).

private

A private interface does not forward any traffic to any other port that is also designated as a private interface. The traffic is blocked unconditionally so no Ethernet frames will be forwarded, including ARP packets. If traffic needs to be selectively blocked, a firewall should be used instead.

span

A span port transmits a copy of every Ethernet frame received by the bridge. The number of span ports configured on a bridge is unlimited, but if an interface is designated as a span port, it cannot also be used as a regular bridge port. This is most useful for snooping a bridged network passively on another host connected to one of the span ports of the bridge. For example, to send a copy of all frames out the interface named fxp4:

```
# ifconfig bridge0 span fxp4
```

sticky

If a bridge member interface is marked as sticky, dynamically learned address entries are treated as static entries in the forwarding cache. Sticky entries are never aged out of the cache or replaced, even if the address is seen on a different interface. This gives the benefit of static address entries without the need to pre-populate the forwarding table. Clients learned on a particular segment of the bridge cannot roam to another segment.

An example of using sticky addresses is to combine the bridge with VLANs in order to isolate customer networks without wasting IP address space. Consider that `CustomerA` is on `vlan100`, `CustomerB` is on `vlan101`, and the bridge has the address `192.168.0.1`:

```
# ifconfig bridge0 addm vlan100 sticky vlan100 addm vlan101 sticky vlan101
# ifconfig bridge0 inet 192.168.0.1/24
```

In this example, both clients see `192.168.0.1` as their default gateway. Since the bridge cache is sticky, one host cannot spoof the MAC address of the other customer in order to intercept their traffic.

Any communication between the VLANs can be blocked using a firewall or, as seen in this example, private interfaces:

```
# ifconfig bridge0 private vlan100 private vlan101
```

The customers are completely isolated from each other and the full `/24` address range can be allocated without subnetting.

The number of unique source MAC addresses behind an interface can be limited. Once the limit is reached, packets with unknown source addresses are dropped until an existing host cache entry expires or is removed.

The following example sets the maximum number of Ethernet devices for `CustomerA` on `vlan100` to 10:

```
# ifconfig bridge0 ifmaxaddr vlan100 10
```

Bridge interfaces also support monitor mode, where the packets are discarded after bpf(4) processing and are not processed or forwarded further. This can be used to multiplex the input of two or more interfaces into a single bpf(4) stream. This is useful for reconstructing the traffic for network taps that transmit the RX/TX signals out through two separate interfaces. For example, to read the input from four network interfaces as one stream:

```
# ifconfig bridge0 addm fxp0 addm fxp1 addm fxp2 addm fxp3 monitor up
# tcpdump -i bridge0
```

35.8.4. SNMP Monitoring

The bridge interface and STP parameters can be monitored via bsnmpd(1) which is included in the FreeBSD base system. The exported bridge MIBs conform to IETF standards so any SNMP client or monitoring package can be used to retrieve the data.

To enable monitoring on the bridge, uncomment this line in /etc/snmpd.config by removing the beginning `#` symbol:

```
begemotSnmpdModulePath."bridge" = "/usr/lib/snmp_bridge.so"
```

Other configuration settings, such as community names and access lists, may need to be modified in this file. See bsnmpd(1) and snmp_bridge(3) for more information. Once these edits are saved, add this line to /etc/rc.conf:

```
bsnmpd_enable="YES"
```

Then, start bsnmpd(1):

```
# service bsnmpd start
```

The following examples use the Net-SNMP software (net-mgmt/net-snmp) to query a bridge from a client system. The net-mgmt/bsnmptools port can also be used. From the SNMP client which is running Net-SNMP, add the following lines to $HOME/.snmp/snmp.conf in order to import the bridge MIB definitions:

```
mibdirs +/usr/share/snmp/mibs
mibs +BRIDGE-MIB:RSTP-MIB:BEGEMOT-MIB:BEGEMOT-BRIDGE-MIB
```

To monitor a single bridge using the IETF BRIDGE-MIB (RFC4188):

```
% snmpwalk -v 2c -c public bridge1.example.com mib-2.dot1dBridge
BRIDGE-MIB::dot1dBaseBridgeAddress.0 = STRING: 66:fb:9b:6e:5c:44
BRIDGE-MIB::dot1dBaseNumPorts.0 = INTEGER: 1 ports
BRIDGE-MIB::dot1dStpTimeSinceTopologyChange.0 = Timeticks: (189959) 0:31:39.59 centi-
```

```
seconds
BRIDGE-MIB::dot1dStpTopChanges.0 = Counter32: 2
BRIDGE-MIB::dot1dStpDesignatedRoot.0 = Hex-STRING: 80 00 00 01 02 4B D4 50
...
BRIDGE-MIB::dot1dStpPortState.3 = INTEGER: forwarding(5)
BRIDGE-MIB::dot1dStpPortEnable.3 = INTEGER: enabled(1)
BRIDGE-MIB::dot1dStpPortPathCost.3 = INTEGER: 200000
BRIDGE-MIB::dot1dStpPortDesignatedRoot.3 = Hex-STRING: 80 00 00 01 02 4B D4 50
BRIDGE-MIB::dot1dStpPortDesignatedCost.3 = INTEGER: 0
BRIDGE-MIB::dot1dStpPortDesignatedBridge.3 = Hex-STRING: 80 00 00 01 02 4B D4 50
BRIDGE-MIB::dot1dStpPortDesignatedPort.3 = Hex-STRING: 03 80
BRIDGE-MIB::dot1dStpPortForwardTransitions.3 = Counter32: 1
RSTP-MIB::dot1dStpVersion.0 = INTEGER: rstp(2)
```

The `dot1dStpTopChanges.0` value is two, indicating that the STP bridge topology has changed twice. A topology change means that one or more links in the network have changed or failed and a new tree has been calculated. The `dot1dStpTimeSinceTopologyChange.0` value will show when this happened.

To monitor multiple bridge interfaces, the private BEGEMOT-BRIDGE-MIB can be used:

```
% snmpwalk -v 2c -c public bridge1.example.com
enterprises.fokus.begemot.begemotBridge
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseName."bridge0" = STRING: bridge0
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseName."bridge2" = STRING: bridge2
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseAddress."bridge0" = STRING: e:ce:3b:5a:9e:13
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseAddress."bridge2" = STRING: 12:5e:4d:74:d:fc
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseNumPorts."bridge0" = INTEGER: 1
BEGEMOT-BRIDGE-MIB::begemotBridgeBaseNumPorts."bridge2" = INTEGER: 1
...
BEGEMOT-BRIDGE-MIB::begemotBridgeStpTimeSinceTopologyChange."bridge0" = Timeticks:
(116927) 0:19:29.27 centi-seconds
BEGEMOT-BRIDGE-MIB::begemotBridgeStpTimeSinceTopologyChange."bridge2" = Timeticks:
(82773) 0:13:47.73 centi-seconds
BEGEMOT-BRIDGE-MIB::begemotBridgeStpTopChanges."bridge0" = Counter32: 1
BEGEMOT-BRIDGE-MIB::begemotBridgeStpTopChanges."bridge2" = Counter32: 1
BEGEMOT-BRIDGE-MIB::begemotBridgeStpDesignatedRoot."bridge0" = Hex-STRING: 80 00 00 40
95 30 5E 31
BEGEMOT-BRIDGE-MIB::begemotBridgeStpDesignatedRoot."bridge2" = Hex-STRING: 80 00 00 50
8B B8 C6 A9
```

To change the bridge interface being monitored via the `mib-2.dot1dBridge` subtree:

```
% snmpset -v 2c -c private bridge1.example.com
BEGEMOT-BRIDGE-MIB::begemotBridgeDefaultBridgeIf.0 s bridge2
```

35.9. Link Aggregation and Failover

FreeBSD provides the lagg(4) interface which can be used to aggregate multiple network interfaces into one virtual interface in order to provide failover and link aggregation. Failover allows traffic to continue to flow as long as at least one aggregated network interface has an established link. Link aggregation works best on switches which support LACP, as this protocol distributes traffic bi-directionally while responding to the failure of individual links.

The aggregation protocols supported by the lagg interface determine which ports are used for outgoing traffic and whether or not a specific port accepts incoming traffic. The following protocols are supported by lagg(4):

failover

This mode sends and receives traffic only through the master port. If the master port becomes unavailable, the next active port is used. The first interface added to the virtual interface is the master port and all subsequently added interfaces are used as failover devices. If failover to a non-master port occurs, the original port becomes master once it becomes available again.

loadbalance

This provides a static setup and does not negotiate aggregation with the peer or exchange frames to monitor the link. If the switch supports LACP, that should be used instead.

lacp

The IEEE® 802.3ad Link Aggregation Control Protocol (LACP) negotiates a set of aggregable links with the peer into one or more Link Aggregated Groups (LAGs). Each LAG is composed of ports of the same speed, set to full-duplex operation, and traffic is balanced across the ports in the LAG with the greatest total speed. Typically, there is only one LAG which contains all the ports. In the event of changes in physical connectivity, LACP will quickly converge to a new configuration.

LACP balances outgoing traffic across the active ports based on hashed protocol header information and accepts incoming traffic from any active port. The hash includes the Ethernet source and destination address and, if available, the VLAN tag, and the IPv4 or IPv6 source and destination address.

roundrobin

This mode distributes outgoing traffic using a round-robin scheduler through all active ports and accepts incoming traffic from any active port. Since this mode violates Ethernet frame ordering, it should be used with caution.

broadcast

This mode sends outgoing traffic to all ports configured on the lagg interface, and receives frames on any port.

35.9.1. Configuration Examples

This section demonstrates how to configure a Cisco® switch and a FreeBSD system for LACP load balancing. It then shows how to configure two Ethernet interfaces in failover mode as well as how to configure failover mode between an Ethernet and a wireless interface.

Example 44. LACP Aggregation with a Cisco® Switch

This example connects two fxp(4) Ethernet interfaces on a FreeBSD machine to the first two Ethernet ports on a Cisco® switch as a single load balanced and fault tolerant link. More interfaces can be added to increase throughput and fault tolerance. Replace the names of the Cisco® ports, Ethernet devices, channel group number, and IP address shown in the example to match the local configuration.

Frame ordering is mandatory on Ethernet links and any traffic between two stations always flows over the same physical link, limiting the maximum speed to that of one interface. The transmit algorithm attempts to use as much information as it can to distinguish different traffic flows and balance the flows across the available interfaces.

On the Cisco® switch, add the *FastEthernet0/1* and *FastEthernet0/2* interfaces to channel group *1*:

```
interface FastEthernet0/1
 channel-group 1 mode active
 channel-protocol lacp
!
interface FastEthernet0/2
 channel-group 1 mode active
 channel-protocol lacp
```

On the FreeBSD system, create the lagg(4) interface using the physical interfaces *fxp0* and *fxp1* and bring the interfaces up with an IP address of *10.0.0.3/24*:

```
# ifconfig fxp0 up
# ifconfig fxp1 up
# ifconfig lagg0 create
# ifconfig lagg0 up laggproto lacp laggport fxp0 laggport fxp1 10.0.0.3/24
```

Next, verify the status of the virtual interface:

```
# ifconfig lagg0
lagg0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        ether 00:05:5d:71:8d:b8
        inet 10.0.0.3 netmask 0xffffff00 broadcast 10.0.0.255
        media: Ethernet autoselect
        status: active
        laggproto lacp
        laggport: fxp1 flags=1c<ACTIVE,COLLECTING,DISTRIBUTING>
        laggport: fxp0 flags=1c<ACTIVE,COLLECTING,DISTRIBUTING>
```

Ports marked as `ACTIVE` are part of the LAG that has been negotiated with the remote switch. Traffic will be transmitted and received through these active ports. Add `-v` to the above command to view the LAG identifiers.

To see the port status on the Cisco® switch:

```
switch# show lacp neighbor
Flags:  S - Device is requesting Slow LACPDUs
        F - Device is requesting Fast LACPDUs
        A - Device is in Active mode       P - Device is in Passive mode

Channel group 1 neighbors

Partner's information:

                  LACP port                        Oper    Port     Port
Port      Flags   Priority  Dev ID          Age    Key     Number   State
Fa0/1     SA      32768     0005.5d71.8db8  29s    0x146   0x3      0x3D
Fa0/2     SA      32768     0005.5d71.8db8  29s    0x146   0x4      0x3D
```

For more detail, type `show lacp neighbor detail`.

To retain this configuration across reboots, add the following entries to /etc/rc.conf on the FreeBSD system:

```
ifconfig_fxp0="up"
ifconfig_fxp1="up"
cloned_interfaces="lagg0"
ifconfig_lagg0="laggproto lacp laggport fxp0 laggport fxp1 10.0.0.3/24"
```

Example 45. Failover Mode

Failover mode can be used to switch over to a secondary interface if the link is lost on the master interface. To configure failover, make sure that the underlying physical interfaces are up, then create the lagg(4) interface. In this example, *fxp0* is the master interface, *fxp1* is the secondary interface, and the virtual interface is assigned an IP address of *10.0.0.15/24*:

```
# ifconfig fxp0 up
# ifconfig fxp1 up
# ifconfig lagg0 create
# ifconfig lagg0 up laggproto failover laggport fxp0 laggport fxp1 10.0.0.15/24
```

The virtual interface should look something like this:

```
# ifconfig lagg0
lagg0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        ether 00:05:5d:71:8d:b8
        inet 10.0.0.15 netmask 0xffffff00 broadcast 10.0.0.255
        media: Ethernet autoselect
        status: active
        laggproto failover
        laggport: fxp1 flags=0<>
        laggport: fxp0 flags=5<MASTER,ACTIVE>
```

Traffic will be transmitted and received on *fxp0*. If the link is lost on *fxp0*, *fxp1* will become the active link. If the link is restored on the master interface, it will once again become the active link.

To retain this configuration across reboots, add the following entries to /etc/rc.conf:

```
ifconfig_fxp0="up"
ifconfig_fxp1="up"
cloned_interfaces="lagg0"
ifconfig_lagg0="laggproto failover laggport fxp0 laggport fxp1 10.0.0.15/24"
```

Example 46. Failover Mode Between Ethernet and Wireless Interfaces

For laptop users, it is usually desirable to configure the wireless device as a secondary which is only used when the Ethernet connection is not available. With lagg(4), it is possible to configure a failover which prefers the Ethernet connection for both performance and security reasons, while maintaining the ability to transfer data over the wireless connection.

This is achieved by overriding the Ethernet interface's MAC address with that of the wireless interface.

> In theory, either the Ethernet or wireless MAC address can be changed to match the other. However, some popular wireless interfaces lack support for overriding the MAC address. We therefore recommend overriding the Ethernet MAC address for this purpose.

> If the driver for the wireless interface is not loaded in the `GENERIC` or custom kernel, and the computer is running FreeBSD 12.1, load the corresponding .ko in /boot/loader.conf by adding `driver_load="YES"` to that file and rebooting. Another, better way is to load the driver in /etc/rc.conf by adding it to `kld_list` (see rc.conf(5) for details) in that file and rebooting. This is needed because otherwise the driver is not loaded yet at the time the lagg(4) interface is set up.

In this example, the Ethernet interface, *re0*, is the master and the wireless interface, *wlan0*, is the failover. The *wlan0* interface was created from the *ath0* physical wireless interface, and the Ethernet interface will be configured with the MAC address of the wireless interface. First, bring the wireless interface up (replacing *FR* with the local 2-letter country code), but do not set an IP address. Replace *wlan0* to match the system's wireless interface name:

```
# ifconfig wlan0 create wlandev ath0 country FR ssid my_router up
```

Determine the MAC address of the wireless interface like this:

```
# ifconfig wlan0
wlan0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
    ether b8:ee:65:5b:32:59
    groups: wlan
    ssid Bbox-A3BD2403 channel 6 (2437 MHz 11g ht/20) bssid 00:37:b7:56:4b:60
    regdomain ETSI country FR indoor ecm authmode WPA2/802.11i privacy ON
    deftxkey UNDEF AES-CCM 2:128-bit txpower 30 bmiss 7 scanvalid 60
    protmode CTS ampdulimit 64k ampdudensity 8 shortgi -stbctx stbcrx
    -ldpc wme burst roaming MANUAL
    media: IEEE 802.11 Wireless Ethernet MCS mode 11ng
    status: associated
    nd6 options=29<PERFORMNUD,IFDISABLED,AUTO_LINKLOCAL>
```

The `ether` line will contain the MAC address of the specified interface. Now, change the MAC address of the Ethernet interface to match:

```
# ifconfig re0 ether b8:ee:65:5b:32:59
```

Make sure the *re0* interface is up, then create the lagg(4) interface with *re0* as master with failover to *wlan0*:

```
# ifconfig re0 up
# ifconfig lagg0 create
# ifconfig lagg0 up laggproto failover laggport re0 laggport wlan0
```

The virtual interface should look something like this:

```
# ifconfig lagg0
lagg0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> metric 0 mtu 1500
        options=8<VLAN_MTU>
        ether b8:ee:65:5b:32:59
        laggproto failover lagghash l2,l3,l4
        laggport: re0 flags=5<MASTER,ACTIVE>
        laggport: wlan0 flags=0<>
        groups: lagg
        media: Ethernet autoselect
        status: active
```

Then, start the DHCP client to obtain an IP address:

```
# dhclient lagg0
```

To retain this configuration across reboots, add the following entries to /etc/rc.conf:

```
ifconfig_re0="ether b8:ee:65:5b:32:59"
wlans_ath0="wlan0"
ifconfig_wlan0="WPA"
create_args_wlan0="country FR"
cloned_interfaces="lagg0"
ifconfig_lagg0="up laggproto failover laggport re0 laggport wlan0 DHCP"
```

35.10. Diskless Operation with PXE

The Intel® Preboot eXecution Environment (PXE) allows an operating system to boot over the network. For example, a FreeBSD system can boot over the network and operate without a local disk, using file systems mounted from an NFS server. PXE support is usually available in the BIOS. To use PXE when the machine starts, select the `Boot from network` option in the BIOS setup or type a function key during system initialization.

In order to provide the files needed for an operating system to boot over the network, a PXE setup also requires properly configured DHCP, TFTP, and NFS servers, where:

- Initial parameters, such as an IP address, executable boot filename and location, server name, and root path are obtained from the DHCP server.
- The operating system loader file is booted using TFTP.
- The file systems are loaded using NFS.

When a computer PXE boots, it receives information over DHCP about where to obtain the initial boot loader file. After the host computer receives this information, it downloads the boot loader via TFTP and then executes the boot loader. In FreeBSD, the boot loader file is /boot/pxeboot. After /boot/pxeboot executes, the FreeBSD kernel is loaded and the rest of the FreeBSD bootup sequence proceeds, as described in The FreeBSD Booting Process.

For UEFI PXE based boot, the actual boot loader file to use is /boot/loader.efi. See the below section Debugging PXE Problems on how to use /boot/loader.efi.

This section describes how to configure these services on a FreeBSD system so that other systems can PXE boot into FreeBSD. Refer to diskless(8) for more information.

As described, the system providing these services is insecure. It should live in a protected area of a network and be untrusted by other hosts.

35.10.1. Setting Up the PXE Environment

The steps shown in this section configure the built-in NFS and TFTP servers. The next section demonstrates how to install and configure the DHCP server. In this example, the directory which will contain the files used by PXE users is /b/tftpboot/FreeBSD/install. It is important that this directory exists and that the same directory name is set in both /etc/inetd.conf and /usr/local/etc/dhcpd.conf.

> The command examples below assume use of the sh(1) shell. csh(1) and tcsh(1) users will need to start a sh(1) shell or adapt the commands to csh(1) syntax.

1. Create the root directory which will contain a FreeBSD installation to be NFS mounted:

   ```
   # export NFSROOTDIR=/b/tftpboot/FreeBSD/install
   # mkdir -p ${NFSROOTDIR}
   ```

2. Enable the NFS server by adding this line to /etc/rc.conf:

   ```
   nfs_server_enable="YES"
   ```

3. Export the diskless root directory via NFS by adding the following to /etc/exports:

   ```
   /b -ro -alldirs -maproot=root
   ```

4. Start the NFS server:

   ```
   # service nfsd start
   ```

5. Enable inetd(8) by adding the following line to /etc/rc.conf:

   ```
   inetd_enable="YES"
   ```

6. Uncomment the following line in /etc/inetd.conf by making sure it does not start with a `#` symbol:

   ```
   tftp dgram udp wait root /usr/libexec/tftpd tftpd blocksize 1468 -l -s /b/tftpboot
   ```

   > The specified tftp blocksize, e.g. 1468 bytes, replaces the default size 512 bytes. Some PXE versions require the TCP version of TFTP. In this case, uncomment the second `tftp` line which contains `stream tcp`.

7. Start inetd(8):

```
# service inetd start
```

8. Install the base system into ${NFSROOTDIR}, either by decompressing the official archives or by rebuilding the FreeBSD kernel and userland (refer to "Updating FreeBSD from Source" for more detailed instructions, but do not forget to add `DESTDIR=${NFSROOTDIR}` when running the `make installkernel` and `make installworld` commands.
9. Test that the TFTP server works and can download the boot loader which will be obtained via PXE:

```
# tftp localhost
tftp> get FreeBSD/install/boot/pxeboot
Received 264951 bytes in 0.1 seconds
```

10. Edit ${NFSROOTDIR}/etc/fstab and create an entry to mount the root file system over NFS:

```
# Device                                        Mountpoint    FSType   Options
Dump Pass
myhost.example.com:/b/tftpboot/FreeBSD/install      /         nfs      ro        0
0
```

Replace *myhost.example.com* with the hostname or IP address of the NFS server. In this example, the root file system is mounted read-only in order to prevent NFS clients from potentially deleting the contents of the root file system.

11. Set the root password in the PXE environment for client machines which are PXE booting :

```
# chroot ${NFSROOTDIR}
# passwd
```

12. If needed, enable ssh(1) root logins for client machines which are PXE booting by editing ${NFSROOTDIR}/etc/ssh/sshd_config and enabling `PermitRootLogin`. This option is documented in sshd_config(5).
13. Perform any other needed customizations of the PXE environment in ${NFSROOTDIR}. These customizations could include things like installing packages or editing the password file with vipw(8).

When booting from an NFS root volume, /etc/rc detects the NFS boot and runs /etc/rc.initdiskless. In this case, /etc and /var need to be memory backed file systems so that these directories are writable but the NFS root directory is read-only:

```
# chroot ${NFSROOTDIR}
# mkdir -p conf/base
# tar -c -v -f conf/base/etc.cpio.gz --format cpio --gzip etc
# tar -c -v -f conf/base/var.cpio.gz --format cpio --gzip var
```

When the system boots, memory file systems for /etc and /var will be created and mounted and the contents of the cpio.gz files will be copied into them. By default, these file systems have a maximum capacity of 5 megabytes. If the archives do not fit, which is usually the case for /var when binary packages have been installed, request a larger size by putting the number of 512 byte sectors needed (e.g., 5 megabytes is 10240 sectors) in ${NFSROOTDIR}/conf/base/etc/md_size and ${NFSROOTDIR}/conf/base/var/md_size files for /etc and /var file systems respectively.

35.10.2. Configuring the DHCP Server

The DHCP server does not need to be the same machine as the TFTP and NFS server, but it needs to be accessible in the network.

DHCP is not part of the FreeBSD base system but can be installed using the net/isc-dhcp44-server port or package.

Once installed, edit the configuration file, /usr/local/etc/dhcpd.conf. Configure the `next-server`, `filename`, and `root-path` settings as seen in this example:

```
subnet 192.168.0.0 netmask 255.255.255.0 {
   range 192.168.0.2 192.168.0.3 ;
   option subnet-mask 255.255.255.0 ;
   option routers 192.168.0.1 ;
   option broadcast-address 192.168.0.255 ;
   option domain-name-servers 192.168.35.35, 192.168.35.36 ;
   option domain-name "example.com";

   # IP address of TFTP server
   next-server 192.168.0.1 ;

   # path of boot loader obtained via tftp
   filename "FreeBSD/install/boot/pxeboot" ;

   # pxeboot boot loader will try to NFS mount this directory for root FS
   option root-path "192.168.0.1:/b/tftpboot/FreeBSD/install/" ;

}
```

The `next-server` directive is used to specify the IP address of the TFTP server.

The `filename` directive defines the path to /boot/pxeboot. A relative filename is used, meaning that /b/tftpboot is not included in the path.

The `root-path` option defines the path to the NFS root file system.

Once the edits are saved, enable DHCP at boot time by adding the following line to /etc/rc.conf:

```
dhcpd_enable="YES"
```

Then start the DHCP service:

```
# service isc-dhcpd start
```

35.10.3. Debugging PXE Problems

Once all of the services are configured and started, PXE clients should be able to automatically load FreeBSD over the network. If a particular client is unable to connect, when that client machine boots up, enter the BIOS configuration menu and confirm that it is set to boot from the network.

This section describes some troubleshooting tips for isolating the source of the configuration problem should no clients be able to PXE boot.

1. Use the net/wireshark package or port to debug the network traffic involved during the PXE booting process, which is illustrated in the diagram below.

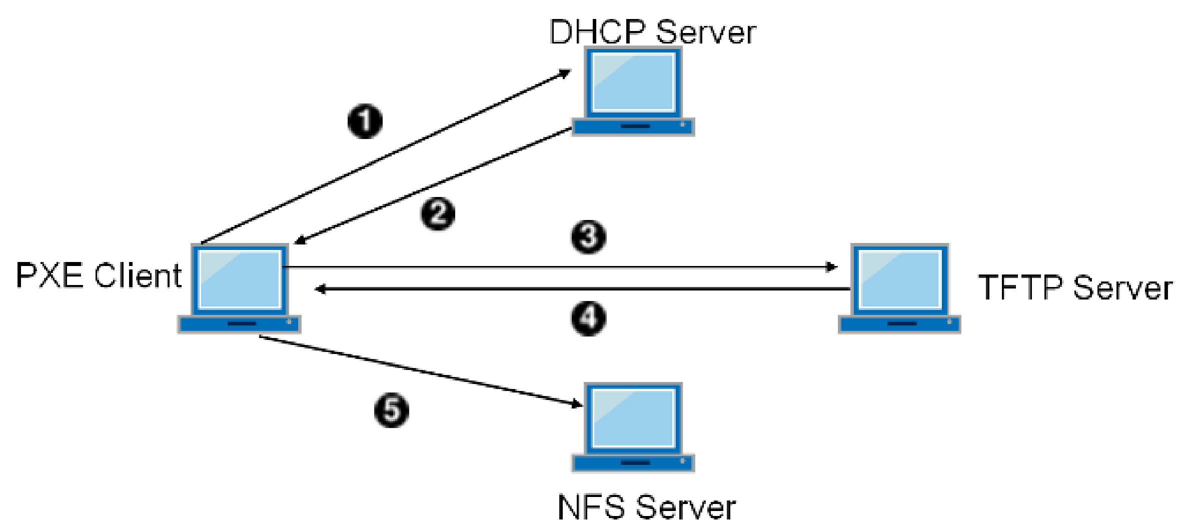
   

   Figure 78. PXE Booting Process with NFS Root Mount

   1. Client broadcasts a DHCPDISCOVER message.
   2. The DHCP server responds with the IP address, next-server, filename, and root-path values.
   3. The client sends a TFTP request to next-server, asking to retrieve filename.
   4. The TFTP server responds and sends filename to client.
   5. The client executes filename, which is pxeboot(8), which then loads the kernel. When the kernel executes, the root file system specified by root-path is mounted over NFS.
2. On the TFTP server, read /var/log/xferlog to ensure that pxeboot is being retrieved from the correct location. To test this example configuration:

   ```
   # tftp 192.168.0.1
   ```

```
tftp> get FreeBSD/install/boot/pxeboot
Received 264951 bytes in 0.1 seconds
```

The `BUGS` sections in tftpd(8) and tftp(1) document some limitations with TFTP.

3. Make sure that the root file system can be mounted via NFS. To test this example configuration:

```
# mount -t nfs 192.168.0.1:/b/tftpboot/FreeBSD/install /mnt
```

4. For UEFI PXE based booting, replace the boot/pxeboot file with the boot/loader.efi file:

```
# chroot ${NFSROOTDIR}
# mv boot/pxeboot boot/pxeboot.original
# cp boot/loader.efi boot/pxeboot
```

35.11. Common Address Redundancy Protocol (CARP)

The Common Address Redundancy Protocol (CARP) allows multiple hosts to share the same IP address and Virtual Host ID (VHID) in order to provide *high availability* for one or more services. This means that one or more hosts can fail, and the other hosts will transparently take over so that users do not see a service failure.

In addition to the shared IP address, each host has its own IP address for management and configuration. All of the machines that share an IP address have the same VHID. The VHID for each virtual IP address must be unique across the broadcast domain of the network interface.

High availability using CARP is built into FreeBSD, though the steps to configure it vary slightly depending upon the FreeBSD version. This section provides the same example configuration for versions before and equal to or after FreeBSD 10.

This example configures failover support with three hosts, all with unique IP addresses, but providing the same web content. It has two different masters named `hosta.example.org` and `hostb.example.org`, with a shared backup named `hostc.example.org`.

These machines are load balanced with a Round Robin DNS configuration. The master and backup machines are configured identically except for their hostnames and management IP addresses. These servers must have the same configuration and run the same services. When the failover occurs, requests to the service on the shared IP address can only be answered correctly if the backup server has access to the same content. The backup machine has two additional CARP interfaces, one for each of the master content server's IP addresses. When a failure occurs, the backup server will pick up the failed master machine's IP address.

35.11.1. Using CARP

Enable boot-time support for CARP by adding an entry for the carp.ko kernel module in

/boot/loader.conf:

```
carp_load="YES"
```

To load the module now without rebooting:

```
# kldload carp
```

For users who prefer to use a custom kernel, include the following line in the custom kernel configuration file and compile the kernel as described in Configuring the FreeBSD Kernel:

```
device  carp
```

The hostname, management IP address and subnet mask, shared IP address, and VHID are all set by adding entries to /etc/rc.conf. This example is for `hosta.example.org`:

```
hostname="hosta.example.org"
ifconfig_em0="inet 192.168.1.3 netmask 255.255.255.0"
ifconfig_em0_alias0="inet vhid 1 pass testpass alias 192.168.1.50/32"
```

The next set of entries are for `hostb.example.org`. Since it represents a second master, it uses a different shared IP address and VHID. However, the passwords specified with `pass` must be identical as CARP will only listen to and accept advertisements from machines with the correct password.

```
hostname="hostb.example.org"
ifconfig_em0="inet 192.168.1.4 netmask 255.255.255.0"
ifconfig_em0_alias0="inet vhid 2 pass testpass alias 192.168.1.51/32"
```

The third machine, `hostc.example.org`, is configured to handle failover from either master. This machine is configured with two CARPVHIDs, one to handle the virtual IP address for each of the master hosts. The CARP advertising skew, `advskew`, is set to ensure that the backup host advertises later than the master, since `advskew` controls the order of precedence when there are multiple backup servers.

```
hostname="hostc.example.org"
ifconfig_em0="inet 192.168.1.5 netmask 255.255.255.0"
ifconfig_em0_alias0="inet vhid 1 advskew 100 pass testpass alias 192.168.1.50/32"
ifconfig_em0_alias1="inet vhid 2 advskew 100 pass testpass alias 192.168.1.51/32"
```

Having two CARPVHIDs configured means that `hostc.example.org` will notice if either of the master servers becomes unavailable. If a master fails to advertise before the backup server, the backup server will pick up the shared IP address until the master becomes available again.

If the original master server becomes available again, `hostc.example.org` will not release the virtual IP address back to it automatically. For this to happen, preemption has to be enabled. The feature is disabled by default, it is controlled via the sysctl(8) variable `net.inet.carp.preempt`. The administrator can force the backup server to return the IP address to the master:

```
# ifconfig em0 vhid 1 state backup
```

Once the configuration is complete, either restart networking or reboot each system. High availability is now enabled.

CARP functionality can be controlled via several sysctl(8) variables documented in the carp(4) manual pages. Other actions can be triggered from CARP events by using devd(8).

35.12. VLANs

VLANs are a way of virtually dividing up a network into many different subnetworks, also referred to as segmenting. Each segment will have its own broadcast domain and be isolated from other VLANs.

On FreeBSD, VLANs must be supported by the network card driver. To see which drivers support vlans, refer to the vlan(4) manual page.

When configuring a VLAN, a couple pieces of information must be known. First, which network interface? Second, what is the VLAN tag?

To configure VLANs at run time, with a NIC of `em0` and a VLAN tag of `5` the command would look like this:

```
# ifconfig em0.5 create vlan 5 vlandev em0 inet 192.168.20.20/24
```

See how the interface name includes the NIC driver name and the VLAN tag, separated by a period? This is a best practice to make maintaining the VLAN configuration easy when many VLANs are present on a machine.

When defining VLANs, ensure that the parent network interface is also configured and enabled. The minimum configuration for the above example would be:

```
# ifconfig em0 up
```

To configure VLANs at boot time, /etc/rc.conf must be updated. To duplicate the configuration above, the following will need to be added:

```
vlans_em0="5"
```

```
ifconfig_em0_5="inet 192.168.20.20/24"
```

Additional VLANs may be added, by simply adding the tag to the `vlans_em0` field and adding an additional line configuring the network on that VLAN tag's interface.

> When defining VLANs in /etc/rc.conf, make sure that the parent network interface is configured and enabled as well. The minimum configuration for the above example would be:
>
> ```
> ifconfig_em0="up"
> ```

It is useful to assign a symbolic name to an interface so that when the associated hardware is changed, only a few configuration variables need to be updated. For example, security cameras need to be run over VLAN 1 on `em0`. Later, if the `em0` card is replaced with a card that uses the ixgb(4) driver, all references to `em0.1` will not have to change to `ixgb0.1`.

To configure VLAN `5`, on the NIC `em0`, assign the interface name `cameras`, and assign the interface an IP address of `192.168.20.20` with a `24`-bit prefix, use this command:

```
# ifconfig em0.5 create vlan 5 vlandev em0 name cameras inet 192.168.20.20/24
```

For an interface named `video`, use the following:

```
# ifconfig video.5 create vlan 5 vlandev video name cameras inet 192.168.20.20/24
```

To apply the changes at boot time, add the following lines to /etc/rc.conf:

```
vlans_video="cameras"
create_args_cameras="vlan 5"
ifconfig_cameras="inet 192.168.20.20/24"
```

Part V: Appendices

Appendix A: Obtaining FreeBSD

A.1. Mirrors

The official mirrors of the FreeBSD project are made up of many machines operated by the project cluster administrators and behind GeoDNS to direct users to the closest available mirror. Current locations are Australia, Brazil, Japan (two sites), Malaysia, South Africa, Sweden, Taiwan, United States of America (California - two sites, Illinois - two sites, New Jersey, and Washington).

Official mirrors service:

Service Name	Protocols	More information
cgit.FreeBSD.org	https	Web interface for the FreeBSD Git repositories.
docs.FreeBSD.org	https	FreeBSD Documentation Portal.
download.FreeBSD.org	https ftp	Same content as `ftp.FreeBSD.org`, `ftp` is a legacy name; `download.FreeBSD.org` is recommended.
git.FreeBSD.org	git over `https` and `ssh`	More details on using git section.
pkg.FreeBSD.org	pkg(8) over `http` and `https`	Official FreeBSD package repositories used by the pkg(8) program.
vuxml.FreeBSD.org / **www.VuXML.org**	https	FreeBSD Project VuXML web page. `pkg audit` fetches the list of vulnerabilities from this service.
www.FreeBSD.org	https	FreeBSD Website.

All official mirrors support IPv4 and IPv6.

http://ftp-archive.FreeBSD.org is not in the GeoDNS Infrastructure, hosted in only one location (US).

The project is looking for new locations; those willing to sponsor, please reach out to the Cluster Administrators team for more information.

Fastly CDN is used to serve **download.FreeBSD.org** and **pkg.FreeBSD.org**, providing acceleration, caching, and global bandwidth distribution for FreeBSD images and packages.

Mirror list maintained by the community and other companies:

Country	Hostname	Protocols
Australia ✉	ftp.au.FreeBSD.org	http http_v6 rsync rsync_v6
	ftp3.au.FreeBSD.org	http ftp rsync
Brazil ✉	ftp2.br.FreeBSD.org	http rsync rsync_v6

Country	Hostname	Protocols
	ftp3.br.FreeBSD.org	http ftp rsync
Bulgaria ✉	ftp.bg.FreeBSD.org	ftp ftp_v6 rsync rsync_v6
Czech Republic ✉	ftp.cz.FreeBSD.org	http http_v6 rsync rsync_v6
Denmark ✉	ftp.dk.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
Finland ✉	ftp.fi.FreeBSD.org	ftp
France ✉	ftp.fr.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp3.fr.FreeBSD.org	ftp
	ftp6.fr.FreeBSD.org	http ftp rsync
Germany ✉	ftp.de.FreeBSD.org	http http_v6 https https_v6 ftp ftp_v6 rsync rsync_v6
	ftp1.de.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp2.de.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp5.de.FreeBSD.org	ftp ftp_v6
	ftp7.de.FreeBSD.org	http http_v6 ftp ftp_v6
Greece ✉	ftp.gr.FreeBSD.org	http http_v6 ftp ftp_v6
	ftp2.gr.FreeBSD.org	http http_v6 ftp ftp_v6 rsync
Japan ✉	ftp.jp.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp2.jp.FreeBSD.org	ftp rsync rsync_v6
	ftp3.jp.FreeBSD.org	http rsync
	ftp4.jp.FreeBSD.org	ftp
	ftp6.jp.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
Kazakhstan ✉	mirror.ps.kz	http ftp
	mirror.neolabs.kz	http ftp
Korea ✉	ftp.kr.FreeBSD.org	http https ftp rsync
	ftp2.kr.FreeBSD.org	rsync

Country	Hostname	Protocols
Latvia ✉	ftp.lv.FreeBSD.org	http ftp
Netherlands ✉	ftp.nl.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp2.nl.FreeBSD.org	http ftp rsync
	mirror.nl.altushost.com	https
New Zealand ✉	ftp.nz.FreeBSD.org	http ftp
Norway ✉	ftp.no.FreeBSD.org	ftp ftp_v6 rsync rsync_v6
Poland ✉	ftp.pl.FreeBSD.org	http http_v6 ftp rsync rsync_v6
Russia ✉	ftp.ru.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp2.ru.FreeBSD.org	https ftp rsync
Slovenia ✉	ftp.si.FreeBSD.org	http http_v6 ftp ftp_v6
South Africa ✉	ftp2.za.FreeBSD.org	http http_v6 ftp_v6
	ftp4.za.FreeBSD.org	http ftp rsync
Sweden ✉	ftp.se.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	mirror.se.altushost.com	https
Taiwan ✉	ftp4.tw.FreeBSD.org	https ftp rsync
	ftp5.tw.FreeBSD.org	http ftp
Ukraine ✉	ftp.ua.FreeBSD.org	http ftp ftp_v6 rsync rsync_v6
United Kingdom ✉	ftp.uk.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp2.uk.FreeBSD.org	http http_v6 https https_v6 ftp ftp_v6
United States of America ✉	ftp11.FreeBSD.org	http http_v6 ftp ftp_v6 rsync rsync_v6
	ftp14.FreeBSD.org	ftp rsync (Former official tier 1)
	ftp5.FreeBSD.org	http http_v6 ftp ftp_v6

The current list of protocols supported by the community mirrors was last updated on 2022-01-31, and it’s not guaranteed.

A.2. Using Git

A.2.1. Introduction

As of December 2020, FreeBSD uses git as the primary version control system for storing all of FreeBSD's base source code and documentation. As of April 2021, FreeBSD uses git as the only version control system for storing all of FreeBSD's Ports Collection.

> Git is generally a developer tool. Users may prefer to use `freebsd-update` ("FreeBSD Update") to update the FreeBSD base system.

This section demonstrates how to install Git on a FreeBSD system and use it to create a local copy of a FreeBSD source code repository.

A.2.2. Installation

Git can be installed from the Ports Collection, or as a package:

```
# pkg install git
```

A.2.3. Running Git

To fetch a clean copy of the sources into a local directory, use `git clone`. This directory of files is called the *working tree*.

Git uses URLs to designate a repository. There are three different repositories, `src` for the FreeBSD system source code, `doc` for documentation, and `ports` for the FreeBSD Ports Collection. All three are reachable over two different protocols: HTTPS and SSH. For example, the URL `https://git.FreeBSD.org/src.git` specifies the main branch of the `src` repository, using the `https` protocol.

Table 51. FreeBSD Git Repository URL Table

Item	Git URL
Read-only src repo via HTTPS	`https://git.FreeBSD.org/src.git`
Read-only src repo via anon-ssh	`ssh://anongit@git.FreeBSD.org/src.git`
Read-only doc repo via HTTPS	`https://git.FreeBSD.org/doc.git`
Read-only doc repo via anon-ssh	`ssh://anongit@git.FreeBSD.org/doc.git`
Read-only ports repo via HTTPS	`https://git.FreeBSD.org/ports.git`
Read-only ports repo via anon-ssh	`ssh://anongit@git.FreeBSD.org/ports.git`

External mirrors maintained by project members are also available; please refer to the External mirrors section.

To clone a copy of the FreeBSD system source code repository:

```
# git clone -o freebsd https://git.FreeBSD.org/src.git /usr/src
```

The `-o freebsd` option specifies the origin; by convention in the FreeBSD documentation, the origin is assumed to be `freebsd`. Because the initial checkout must download the full branch of the remote repository, it can take a while. Please be patient.

Initially, the working tree contains source code for the `main` branch, which corresponds to CURRENT. To switch to 13-STABLE instead:

```
# cd /usr/src
# git checkout stable/13
```

The working tree can be updated with `git pull`. To update /usr/src created in the example above, use:

```
# cd /usr/src
# git pull --rebase
```

The update is much quicker than a checkout, only transferring files that have changed.

A.2.4. Web-based repository browser

The FreeBSD project uses cgit as the web-based repository browser: https://cgit.FreeBSD.org/.

A.2.5. For Developers

For information about write access to repositories see the Committer's Guide.

A.2.6. External mirrors

Those mirrors are not hosted in FreeBSD.org but still maintained by the project members. Users and developers are welcome to pull or browse repositories on those mirrors. Pull requests for the `doc` and `src` GitHub repositories are being accepted; otherwise, the project workflow with those mirrors is still under discussion.

Codeberg

- doc: https://codeberg.org/FreeBSD/freebsd-doc
- ports: https://codeberg.org/FreeBSD/freebsd-ports
- src: https://codeberg.org/FreeBSD/freebsd-src

GitHub

- doc: https://github.com/freebsd/freebsd-doc
- ports: https://github.com/freebsd/freebsd-ports
- src: https://github.com/freebsd/freebsd-src

GitLab

- doc: https://gitlab.com/FreeBSD/freebsd-doc
- ports: https://gitlab.com/FreeBSD/freebsd-ports
- src: https://gitlab.com/FreeBSD/freebsd-src

A.2.7. Mailing lists

The main mailing list for general usage and questions about git in the FreeBSD project is freebsd-git. For more details, including commit messages lists, see the Mailing Lists chapter.

A.2.8. SSH host keys

- gitrepo.FreeBSD.org host key fingerprints:
  - ECDSA key fingerprint is `SHA256:seWO5D27ySURcx4bknTNKlC1mgai0whP443PAKEvvZA`
  - ED25519 key fingerprint is `SHA256:lNR6i4BEOaaUhmDHBA1WJsO7H3KtvjE2r5q4sOxtIWo`
  - RSA key fingerprint is `SHA256:f453CUEFXEJAXlKeEHV+ajJfeEfx9MdKQUD7lIscnQI`
- git.FreeBSD.org host key fingerprints:
  - ECDSA key fingerprint is `SHA256:/UlirUAsGiitupxmtsn7f9b7zCWd0vCs4Yo/tpVWP9w`
  - ED25519 key fingerprint is `SHA256:y1ljKrKMD3lDObRUG3xJ9gXwEIuqnh306tSyFd1tuZE`
  - RSA key fingerprint is `SHA256:jBe6FQGoH4HjvrIVM23dcnLZk9kmpdezR/CvQzm7rJM`

These are also published as SSHFP records in DNS.

Appendix B: Bibliography

While manual pages provide a definitive reference for individual pieces of the FreeBSD operating system, they seldom illustrate how to put the pieces together to make the whole operating system run smoothly. For this, there is no substitute for a good book or users' manual on UNIX® system administration.

B.1. FreeBSD Bibliography

- **Absolute FreeBSD: The Complete Guide To FreeBSD**, Third Edition, published by No Starch Press, 2018. ISBN: 978-1593278922
- **FreeBSD Mastery: Storage Essentials**, published by Tilted Windmill Press, 2014. ISBN: 978-1642350098
- **FreeBSD Mastery: Specialty Filesystems**, published by Tilted Windmill Press, 2015. ISBN: 978-1642350111
- **FreeBSD Mastery: ZFS**, published by Tilted Windmill Press, 2015. ISBN: 978-1642350005
- **FreeBSD Mastery: Advanced ZFS**, published by Tilted Windmill Press, 2016. ISBN: 978-0692688687
- **FreeBSD Mastery: Jails**, published by Tilted Windmill Press, 2019. ISBN: 978-1642350241
- **FreeBSD Device Drivers: A Guide for the Intrepid**, published by No Starch Press, 2012. ISBN: 978-1593272043
- **The Design And Implementation Of The Freebsd Operating System**, Second Edition, published by Pearson Education, Inc., 2014. ISBN: 978-0321968975
- **UNIX and Linux System Administration Handbook**, Fifth Edition, published by Pearson Education, Inc., 2017. ISBN: 978-0134277554
- **Designing BSD Rootkits**, published by No Starch Press, 2007. ISBN: 978-1593271428
- **FreeBSD Jails using VNETs**, published in gumroad

B.2. Security Reference

- **The Book of PF: A No-Nonsense Guide to the OpenBSD Firewall**, Third Edition, published by No Starch Press, 2014. ISBN: 978-1593275891
- **SSH Mastery: OpenSSH, PuTTY, Tunnels, and Keys**, Second Edition, 2018. ISBN: 978-1642350029

B.3. UNIX® History

- Lion, John *Lion's Commentary on UNIX, 6th Ed. With Source Code*. ITP Media Group, 1996. ISBN 1573980137
- Raymond, Eric S. *The New Hacker's Dictionary, 3rd edition*. MIT Press, 1996. ISBN 0-262-68092-0. Also known as the Jargon File

- Salus, Peter H. *A quarter century of UNIX*. Addison-Wesley Publishing Company, Inc., 1994. ISBN 0-201-54777-5
- Simon Garfinkel, Daniel Weise, Steven Strassmann. *The UNIX-HATERS Handbook*. IDG Books Worldwide, Inc., 1994. ISBN 1-56884-203-1. Out of print, but available online.
- Don Libes, Sandy Ressler *Life with UNIX* - special edition. Prentice-Hall, Inc., 1989. ISBN 0-13-536657-7
- *The BSD family tree*. https://cgit.freebsd.org/src/tree/share/misc/bsd-family-tree or /usr/share/misc/bsd-family-tree on a FreeBSD machine.
- *Networked Computer Science Technical Reports Library*.
- *Old BSD releases from the Computer Systems Research group (CSRG)*. http://www.mckusick.com/csrg/: The 4CD set covers all BSD versions from 1BSD to 4.4BSD and 4.4BSD-Lite2 (but not 2.11BSD, unfortunately). The last disk also holds the final sources plus the SCCS files.
- Kernighan, Brian *Unix: A History and a Memoir*. Kindle Direct Publishing, 2020. ISBN 978-169597855-3

B.4. Periodicals, Journals, and Magazines

- Admin Magazin (in German), published by Medialinx AG. ISSN: 2190-1066
- BSD Now - Video Podcast, published by Jupiter Broadcasting LLC
- FreeBSD Journal, published by S&W Publishing, sponsored by The FreeBSD Foundation. ISBN: 978-0-615-88479-0

Appendix C: Resources on the Internet

Development of FreeBSD is too rapid for print media to be practical for keeping people informed. For awareness of developments: electronic alternatives to print are best.

The FreeBSD user community provides much technical support — with forums, chat and email amongst the most popular and effective means of communication.

The most important points of contact are outlined below. The Community wiki area may be more up-to-date.

Please make the FreeBSD documentation project mailing list aware of any resource that is either redundant, or not yet listed below.

C.1. Websites

- The FreeBSD Forums provide a web based discussion forum for FreeBSD questions and technical discussion.
- The FreeBSD Wiki provides various bits of information that hadn't yet made it into the Handbook.
- The Documentation Portal offers much more than the FreeBSD Handbook alone; there are more than forty books and articles.
- The FreeBSD Journal is a free, professionally-edited, bi-monthly technical magazine released by The FreeBSD Foundation.
- The BSDConferences YouTube Channel provides a collection of high quality videos from BSD conferences around the world. This is a great way to watch key developers give presentations about new work in FreeBSD.
- FreeBSD Status Reports are released every three months and track progress of FreeBSD development.
- There's a FreeBSD-focused Reddit group at r/freebsd.
- Super User and Server Fault, the Stack Exchange services for system administrators.
- FreeBSD Discord server, a communications and community-building service, where FreeBSD community members can socialise, obtain support or support others, learn, contribute, collaborate, and stay up to date with all things FreeBSD-related.
- IRC channels, a widely implemented, technically mature, open standard text chat.

C.2. Mailing Lists

The mailing lists are the most direct way of addressing questions or opening a technical discussion to a concentrated FreeBSD audience. There are a wide variety of lists on a number of different FreeBSD topics. Sending questions to the most appropriate mailing list will invariably assure a faster and more accurate response.

Technical list threads should remain technical.

All users and developers of FreeBSD should subscribe to the FreeBSD announcements mailing list.

> To test FreeBSD mailing list capabilities, aim for the FreeBSD test mailing list. Please do not send test messages to any other list.

When in doubt about what list to post a question to, see How to get best results from the FreeBSD-questions mailing list.

Before posting to any list, please:

- learn about how to best use the mailing lists, such as how to help avoid frequently-repeated discussions, by reading the Mailing List Frequently Asked Questions (FAQ) document
- search the archives, to tell whether someone else has already posted the same.

Archive search interfaces include:

- https://lists.freebsd.org/search (FreeBSD, experimental)
- https://www.freebsd.org/search/ (DuckDuckGo)

Note that this also means that messages sent to FreeBSD mailing lists are archived in perpetuity. When protecting privacy is a concern, consider using a disposable secondary email address and posting only public information.

FreeBSD-provided archives:

- do not present links as links
- do not present inline images
- do not present HTML content of HTML messages.

The FreeBSD public mailing lists can be consulted here.

C.2.1. How to Subscribe or Unsubscribe

At https://lists.freebsd.org, click the name of a list to reveal its options.

To post, after subscribing, send mail to `listname@FreeBSD.org`. The message will be redistributed to list members.

C.2.2. Lists Basic Rules

All FreeBSD mailing lists have certain basic rules which must be adhered to by anyone using them. Failure to comply with these guidelines will result in two (2) written warnings from the FreeBSD Postmaster postmaster@FreeBSD.org, after which, on a third offense, the poster will removed from all FreeBSD mailing lists and filtered from further posting to them. We regret that such rules and measures are necessary at all, but today's Internet is a pretty harsh environment, it would seem, and many fail to appreciate just how fragile some of its mechanisms are.

Rules of the road:

- The topic of any posting should adhere to the basic description of the list it is posted to. If the list is about technical issues, the posting should contain technical discussion. Ongoing irrelevant chatter or flaming only detracts from the value of the mailing list for everyone on it and will not be tolerated. For free-form discussion on no particular topic, the FreeBSD chat mailing list is freely available and should be used instead.
- No posting should be made to more than 2 mailing lists, and only to 2 when a clear and obvious need to post to both lists exists. For most lists, there is already a great deal of subscriber overlap and except for the most esoteric mixes (say "-stable & -scsi"), there really is no reason to post to more than one list at a time. If a message is received with multiple mailing lists on the `Cc` line, trim the `Cc` line before replying. *The person who replies is still responsible for cross-posting, no matter who the originator might have been.*
- Personal attacks and profanity (in the context of an argument) are not allowed, and that includes users and developers alike. Gross breaches of netiquette, like excerpting or reposting private mail when permission to do so was not and would not be forthcoming, are frowned upon but not specifically enforced.
- Advertising of non-FreeBSD related products or services is strictly prohibited and will result in an immediate ban if it is clear that the offender is advertising by spam.

C.2.3. Filtering on the Mailing Lists

The FreeBSD mailing lists are filtered in multiple ways to avoid the distribution of spam, viruses, and other unwanted emails. The filtering actions described in this section do not include all those used to protect the mailing lists.

Only certain types of attachments are allowed on the mailing lists. All attachments with a MIME content type not found in the list below will be stripped before an email is distributed on the mailing lists.

- application/octet-stream
- application/pdf
- application/pgp-signature
- application/x-pkcs7-signature
- message/rfc822
- multipart/alternative
- multipart/related
- multipart/signed
- text/html
- text/plain
- text/x-diff
- text/x-patch

> Some of the mailing lists might allow attachments of other MIME content types, but the above list should be applicable for most of the mailing lists.

If a multi-part message includes text/plain and text/html parts:

- addressees will receive both parts
- lists.freebsd.org will present text/plain with an option to view original text (source, with raw HTML amongst the parts).

If text/plain does not accompany text/html:

- there will be conversion from HTML to plain text.

C.3. Usenet Newsgroups

In addition to two FreeBSD specific newsgroups, there are many others in which FreeBSD is discussed or are otherwise relevant to FreeBSD users.

C.3.1. BSD Specific Newsgroups

- comp.unix.bsd.freebsd.announce
- comp.unix.bsd.freebsd.misc
- de.comp.os.unix.bsd (German)
- fr.comp.os.bsd (French)

C.3.2. Other UNIX® Newsgroups of Interest

- comp.unix
- comp.unix.questions
- comp.unix.admin
- comp.unix.programmer
- comp.unix.shell
- comp.unix.misc
- comp.unix.bsd

C.3.3. X Window System

- comp.windows.x

Appendix D: OpenPGP Keys

The OpenPGP keys of the `FreeBSD.org` officers are shown here. These keys can be used to verify a signature or send encrypted email to one of the officers. A full list of FreeBSD OpenPGP keys is available in the PGP Keys article. The complete keyring can be downloaded at pgpkeys.txt.

D.1. Officers

D.1.1. Security Officer Team `<security-officer@FreeBSD.org>`

```
pub   rsa4096/D9AD2A18057474CB 2022-12-11 [C] [expires: 2030-01-24]
      Key fingerprint = 0BE3 3275 D74C 953C 79F8  1107 D9AD 2A18 0574 74CB
uid                            FreeBSD Security Officer <security-officer@freebsd.org>
sub   rsa4096/6E58DE901F001AEF 2022-12-11 [S] [expires: 2028-01-24]
sub   rsa4096/46DB26D62F6039B7 2022-12-11 [E] [expires: 2028-01-24]
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGOVdeUBEADHF5VGg1iPbACB+7lomX6aDytUf0k2k2Yc/Kp6lfYv7JKU+1nr
TcNF7Gt1YkajPSeWRKNZw/X94g4w5TEOHbJ6QQWx9g+N7RjEq75actQ/r2N5zY4S
ujfFTepbvgR55mLTxlxGKFBmNrfNbpHRyh4GwFRgPlxf5Jy9SB+0m54yFS4QlSd0
pIzO0CLkjHUFy/8S93oSK2zUkgok5gLWruBXom+8VC3OtBElkWswPkE1pKZvMQCv
VyM+7BS+MCFXSdZczDZZoEzpQJGhUYFsdg0KqlLv6z1rP+HsgUYKTkRpcrumDQV0
MMuCE4ECU6nFDDTnbR8Wn3LF5oTt0GtwS0nWf+nZ1SFTDURcSPR4Lp/PKjuDAkOS
P8BaruCNx1ItHSwcnXw0gS4+h8FjtWNZpsawtzjjgApcl+m9KP6dkBcbN+i1DHm6
NG6YQVtVWyN8aOKmoC/FEm1CWh1bv+ri9XOkF2EqT/ktbjbT1hFoFGBkS9/35y1G
3KKyWtwKcyF4OXcArl6sQwGgiYnZEG3sUMaGrwQovRtMf7le3cAYsMkXyiAnEufa
deuabYLD8qp9L/eNo+9aZmhJqQg4EQb+ePH7bGPNDZ+M5oGUwReX857FoWaPhs4L
dAKQ1YwASxdKKh8wnaamjIeZSGP5TCjurH7pADAIaB3/D+ZNl2a7od+C1wARAQAB
tDdGcmVlQlNEIFNlY3VyaXR5IE9mZmljZXIgPHNlY3VyaXR5LW9mZmljZXJAZnJl
ZWJzZC5vcmc+iQJSBBMBCgA8AhsBBAsJCAcEFQoJCAUWAgMBAAIeBQIXgBYhBAvj
MnXXTJU8efgRB9mtKhgFdHTLBQJjlXeQBQkF3u+rAAoJENmtKhgFdHTLOVoQALS3
cj7rqYkHiV4zDYrgPEp9O1kAyGI8VdfGAMkDVTqr+wP4v/o7LIUrgwZl5qxesVFB
VknFr0Wp5g9h0iAjasoI5sDd6tH2SmumhBHXFVdftzDQhrugxH6fWRhHs0SaFYCk
Qt5nFbcpUfWgtQ35XTbsL8iENdYpjKXsSFQrJneGSwxIjWYTFn6ps/AI3gwR8+Bn
OffEFdYugJ049O6Vu6YBFJHrnMO7NbF4v95dVYuLtpMIaXWM+V9KITmhaBzFz5fM
Q7UOzcLlbxOYKNIWcp8QQk429mayKW5VUeUExUD1ZzBHn+P6ZG7QTMDu/RmBqiHo
ewCMVz4n9uXT5BiOngE4CvS0WQwHzK+k9MLpG2u/Bo9+LT0Ceh9Ou1rfU5+0tRwl
GyOFFjf3INS7I7gkcAwxQ7dzDItN/UQPZpg8y9mABU2x4enz0AvTnb61d/1dnTEr
tdNgU433he0ZnD1HurZCjBEWC656wv6iMdWcD8gjhMbmEpPmjvXcYlTO6zhEygSM
DiwdQCWK2W4++YJerA6ULBi3niNWBpofOFH8XylV56ruhjtHCo7+/3carcMoPOJv
lVZ1zCKxLro3TRBT15JTFBGqblRyTopFK3PuxW//GTnZOtpQEOV6yL4RAXcWeC1d
1hb5k/YxUmRF6XsDNEH4b08T8ZO8dV3dAV43Wh1oiQEzBBABCAAdFiEEuyjUCzYO
7pNq7RVv5fe8y6O93fgFAmObXVYACgkQ5fe8y6O93fiBlwf/W8y1XXJIx1ZA3n6u
f7aS70rbP9KFPr4U0dixwKE/gbtIQ9ckeNXrDDWz0v0NCz4qS+33IPiJg1WcY3vR
W90e7QgAueCo5TdZPImPbCs42vadpa5byMXS4Pw+xyT+d/yp2oLKYbj3En4bg1GM
w71DezIjvV+e01UR++u1t9yZ8LOWM5Kumz1zyQLZDZ8qIKt1bBfpa+E0cEqtNQWu
```

```
iGhQE3AHI8eWV+jBkg5y2zHRIevbWb1UPsj43lgkFtAGHk9rrM8Rmgr4AXr531iD
srBwauKZ/MElcF3MINuLH+gkPPaFHw/YIpLRLaZXZVsw3Xi1RNXI2n2ea29dvs/C
Lcf1vYkCMwQQAQgAHRYhBPwOh4rlr+eIAo1jVdOXkvSep+XCBQJjm14FAAoJENOX
kvSep+XC0DcP/1ZB7k9p1T+9QbbZZE1PJiHby3815ccH3XKexbNmmakHIn3L6Cet
F891Kqt9ssbhFRMNtyZ/k/8y8Hv5bKxVep5/HMyK+8aqfDFN0WMrqZh0/CiR6DJh
gnAmPNw/hAVHMHaYGII9kCrFfPFJ02FKoc81g9F08odb7TV+UlvRjkErhRxF+dGS
wQoO0RCbf0Z1cs7nd0Vb2z4IJh4XMxBjWc/uQ2Q9dH/0uRzwpAnR4YX+MG5YrX7Z
zBvDyR0r76iQwRSDKgioNgkr6R3rq1NZGdaj+8b0LzdOqtzKJ/eupDe3+H67e/EN
qymtreGjrubpiU9bKvYArisUqhE5KtguryvR6Qz9bj87nPg33DT3WWGVrwFRxBox
dbWzjQFv0wug8m4GAwVF7fPR5/eW7IHw8zvgn0vSPcZz7MZ4e6Y5jN4kA5/xWJYZ
Sps54qQWB+FA30unIXN68KqdIzONIbtaY3W4/JjJUCm4T+wEjKaH+wJX8w1DMjlg
mkTmGh/UrTyC1vXbPgk9Sy3cRTICR1T9z7W8UlmTtnKrUklrjlFR7SXzrEXzLGOX
Fm+NEHpHNXqzcm6c3QfzY/yQ9HSAQ/t7SUQ9caRePbDz3/msyPxtGFor9roQv6VN
wRXCyRgkH4Y5tPhJAQ8G/FxX+VXFb93QL0lfelb23/BBu6cUwW63SRn5iHUEExYI
AB0WIQQeB2Johg/5/ikUnJwDU9SVF1S13AUCZISO3wAKCRADU9SVF1S13NnqAP95
LA1Om9XSAKl76VtV+L3JPDdAwIdbNa0OsRT4Wm7U3wD/YoFrdHXVHHQFKwYeUUhj
XZcxnZLe9Ixo0/JP+RVFVw6JAlIEEwEKADwCGwEECwkIBwQVCgkIBRYCAwEAAh4F
AheAFiEEC+MydddMlTx5+BEH2a0qGAV0dMsFAmluhFoFCQ1lXf0ACgkQ2a0qGAV0
dMv8ixAAhYn3uxnjTCBZ5BJ6FMdwFXgGZMZVnL1qEoHA7HWB0ZoQlu29sFzBX97x
rkVZfWiU82V9/PYyHh4EY0hN9oOJMDDhSgi+DfdoEablIcIXR9uBBL/Snnak6uEF
gOD6VFH1T3o7iBmS/ixee2YFXq5QxAwEiANSAsxDm2rlyztPc5xM76fz5eEcgp1m
UDI7qFjoEkJVvRraJGVy3AUh4sr/94VAXPuCa7JnC6a8j9SD8XtzqIfcuqNBXXfA
m4ji4fN+KcDHtZPdkNLDBu1DEXQkOR/1S++NmXHYCEdQEuVOwLVsy6vQqgc+unju
Xh9sOrdiXUWylHh2hvgmwkjhMGOREiIRwNoA09m0OKZ2jo2dSlhNdVFRD5yhD92t
X9BarQbbjQtyCvoWnCJMFfw/vDnxfDdUdV3GxLhaOWvR8ItWQ7ts8tEZSjR/G76/
fHj1JRNuHRdhwype6ehieewJSMyzByhT0A3Njv3KLI4nwQj//MYzB5FJ9uOmKLcG
w7PB0yVjW4oRbG62+Qv6IQybCyrAL+12qgbUXbMsTqMwmsmdgtsqr/7OJNo3PBxX
2rUEio/mez8wuzNB0mJtlMynoMqXtbFX7AAXHk16fOGd6L6guW8+IhPedT3TvrjS
hqiQVvjOhM6rEYZ/YUfcIdkWJVpmE6+MRPVUufyMWeHaWqsKXoa5Ag0EY5V2yQEQ
AOqjzPpMUCGu8eElXnAd2PruC6hi+lc/yC90KqizxIuW6qLQBaAkTCWq7suYpDqo
ygn7YM3rL50S285WAECAXrcst/cVAqr0UH/e6p4iJCUIiXcfjd/wq20RnN/+VuvL
hjpCFLY5czfVS31D7Uh9MbC+zUTz8nVTiNCsAao0qSdfJDIzB4nSO+9xIsme/dLs
I5QlU5PdxOBV6HdEhCUXOoratJCbKAOlLxtPwyMKxmv4oZ7Mqlt10peKjhpBb97q
cIzJhHxujQZD0OmzIA6xoQ2eSCGdxCEDsZ09kr3Esw1AwKnQ51xmWpFWNFk6627M
1bo8+hzOz81CrTZhYrgE+1JXv6V1L2A9lMsimdE1BHNycDS+dBOpIB9qxXCwAab4
ykfvNxox/ZPDUrTy7v7mDI5uDNTNCYYsKCj1UidycOKzSziB9Oa2uvmMJ5XstgNB
f7Z8Cky1dtVd4ol6bU9L5nos9tbYeSXFi4bmcWB7AJiVCMq6N+LBbUKWGLglB4TU
1qhttpqv31X9V6ges5gARY/RuRTKsVyhwsn7SDcqmNKRy0im2AYakwEp7hT07ula
hOSLxjP+5hCf+nSJlwbxJ8ozwjjbzeN2yLlJSI00klkIFBNUDt3wzFRW/n6qlf+/
lepgzekfNrYMtfPB8AT07Z2A3U4xlgiV346dZymbY/EjABEBAAGJBHIEGAEKACYW
IQQL4zJ110yVPHn4EQfZrSoYBXR0ywUCY5V2yQIbAgUJAgIpAAJACRDZrSoYBXR0
y8F0IAQZAQoAHRYhBLYVJ36BCH33XIGDO25Y3pAfABrvBQJjlXbJAAoJEG5Y3pAf
ABrvuBsQAOlQFPXhx6whO4yw5ZizIS02YHhSVMVYKS2T9jPIKi1qxnEiEw9eKH0b
W0Oj0TEhZPyM2NJID7DRWK5r8+KsMu8jwm1fUmIrefAx6fCVfCWRECT1MlbL3jhh
6AcX/nK2e3Bn8vgExhzczO3JlvD6wPCc0FkpiY7yDB9ihu1+gbE5Hg6dvftttRXD
rbEdAifbNp9KYxDigxdlOb0S14hjCBysLWH5Su/khcIlkeuqZcI8TmDldnUb2OqT
CVpFhaNwsPSrHBzmb0s2sXo4FL03pLsOdwhi31W6kjk4KvW5FKrOpoEwUMKVNMf5
0DHdvonUoUHRSIc/cV5NqUWHwvc0T5031qk0CCRRa/+/iij/p2RG7c1mx7ZECj+j
ZfmvjSqT+WHJ1BFlNJMWyK4fdVRZWyCaoAecdbukwzDwUCUqHJFIWeFtbut7SOPx
cwg7sbnKNApAKdi491dvH75s/U/OwRYO/2P+ymHlqtyix2jq0ReSVYcQPXswQ8i2
ifX41F+xTSl4RWCBBeXB1Nxk3+HsV4Jnnp1zAJZ0KlKW/oJxbNFdI1TImkpr2p8i
```

```
oFf+aiePLvDkgeaG8vABgjoihPXWHVAMR8Z+GvBY/A6OdexpibkTvC/zDr0/Exs4
lsylZKDwvvFbctcpHVXBeCBQLX5vfLrsTkaCLWF/SV9OdMykvYKU7ZAP+gKEwhp+
HPFuOHZbOBhqFUdkfeCkdzX/QGdzTuz349roRhgz2vRfN7MtbtuzA6NWWhEWt5Dc
UgX/Y5I3Q4Z2bt3JiXQ6WJMgMMOXAr+XxtxyRRykc1HV3DQ/cq8OWYubNnIbgebP
NIFr2OIWKsR9yDaucZzpmLfzaMZUAu5hWmU9fIw5SIKGnQABBnNMhilfD+CkETp6
baTvjTK4rpaobjJdeCTrsWgfXRNC8x3hDvcrjPD70MylOGVQdx8GYChWJnCKXsLT
GX7KwdfxkjclTyzWvdcCemp0eLhamLGb9y1dtWdNIDcVCvZJy0lipHVUdFYYxb4i
LZJANL631tlPM6AA8sO1/L4mqEGnAIHVrUQd+2QkSiOl9mKlpgaR/fJz683BR5Qe
n9ywX0JPtBupqPW3t9VbO/uNxUqlHCeAhPi9NLOpujpYLfgW5QAfS3u0nkp5nrbk
CoQUua2q0Oj7J0mFmtWtcE1c9+THmFJVb8j2G9yQw3ADe3Qp9ALazP5nVDVri8NZ
BhHK1/KuBmRYZtcyfqXUnKoiiWAlm5rHaRiztW7e3wqm2oJu/RkEAagybutEuBWh
2Ej2+gDxjEKKtIKGu54lif4kqTwwjKTcN1ekGihwwgCMUkBSBeNXk1ClkzLFHwES
JCcFwdEgpVYQTKFsu0emYISyco3IpUajGzfUiQRyBBgBCgAmAhsCFiEEC+MydddM
lTx5+BEH2a0qGAV0dMsFAmWFy28FCQPyfaYCQMF0IAQZAQoAHRYhBLYVJ36BCH33
XIGDO25Y3pAfABrvBQJjlXbJAAoJEG5Y3pAfABrvuBsQAOlQFPXhx6whO4yw5Ziz
IS02YHhSVMVYKS2T9jPIKi1qxnEiEw9eKH0bW0Oj0TEhZPyM2NJID7DRWK5r8+Ks
Mu8jwm1fUmIrefAx6fCVfCWRECT1MlbL3jhh6AcX/nK2e3Bn8vgExhzczO3JlvD6
wPCc0FkpiY7yDB9ihu1+gbE5Hg6dvftttRXDrbEdAifbNp9KYxDigxdlOb0S14hj
CBysLWH5Su/khcIlkeuqZcI8TmDldnUb2OqTCVpFhaNwsPSrHBzmb0s2sXo4FL03
pLsOdwhi31W6kjk4KvW5FKrOpoEwUMKVNMf50DHdvonUoUHRSIc/cV5NqUWHwvc0
T5031qk0CCRRa/+/iij/p2RG7c1mx7ZECj+jZfmvjSqT+WHJ1BFlNJMWyK4fdVRZ
WyCaoAecdbukwzDwUCUqHJFIWeFtbut7SOPxcwg7sbnKNApAKdi491dvH75s/U/O
wRYO/2P+ymHlqtyix2jq0ReSVYcQPXswQ8i2ifX41F+xTSl4RWCBBeXB1Nxk3+Hs
V4Jnnp1zAJZ0KlKW/oJxbNFdI1TImkpr2p8ioFf+aiePLvDkgeaG8vABgjoihPXW
HVAMR8Z+GvBY/A6OdexpibkTvC/zDr0/Exs4lsylZKDwvvFbctcpHVXBeCBQLX5v
fLrsTkaCLWF/SV9OdMykvYKUCRDZrSoYBXR0yyOqEACitDvbkbfjaton6izr4T8Q
U2yvhJHkf4B6KeVDbKY1J4784OxXp2bJgPeF53SYBe8gm3YHjp8ULh4A/19U4hsw
yE8ymcm5nIs8OLyBdxkuBZJGEnzxH3woiyYqWH7991kzhEjUkuMgKLuTI1HiO0oL
MuPQNhUHOnWafSVPC0XO/tIL12OmoUuc7ligY9Z9AceFjTZOuHamixHAAc6hpxdI
W+yhC/qTpc2VK0niWeuQfq3453iRTf9MnR5Beztl3ZYRWcx7UiFuKGwZwBibNnNm
Us6GyQcJ5UTa1oeJcLqHi0Lf/r0jXo3wgJq7EZjjVyU+GI2ZVoDOaS6c4/OvLm62
XoeSlnn/dQxUcjUki+x8lb69IxSF1xAgsC/oNtFZYd5rHdlnqIBUYK0lLtSCXBkz
VeivSiQa0hL5on8LDu1nw2bXyW61yt/YxVb4FanMxAqdYVBhOfU0RaPNifH01rbb
4TwC9bTZN1LQ1KI/Swb/SruUE0RyT28fhYRtsReS2PnUODghJSFDJbwFbBZf6RKI
16q1xqKRRvxIWPm+lMOi1NLOKR9P+OKy9HmChMw0UJUcVl1cJ2xtRl3wi5t6AA6H
oNv/TrLeYVgMR9wYmKlpvjTQ5jTdrbHD1XP5jGsp8QsJMGja1m/7cryReCpcVxvI
meReaOdgz+zDmQqq3O5zuIkEcgQYAQoAJgIbAhYhBAvjMnXXTJU8efgRB9mtKhgF
dHTLBQJnhEEJBQkF0/JAAkDBdCAEGQEKAB0WIQS2FSd+gQh991yBgztuWN6QHwAa
7wUCY5V2yQAKCRBuWN6QHwAa77gbEADpUBT14cesITuMsOWYsyEtNmB4UlTFWCkt
k/YzyCotasZxIhMPXih9G1tDo9ExIWT8jNjSSA+w0Viua/PirDLvI8JtX1JiK3nw
MenwlXwlkRAk9TJWy944YegHF/5ytntwZ/L4BMYc3MztyZbw+sDwnNBZKYmO8gwf
YobtfoGxOR4Onb37bbUVw62xHQIn2zafSmMQ4oMXZTm9EteIYwgcrC1h+Urv5IXC
JZHrqmXCPE5g5XZ1G9jqkwlaRYWjcLD0qxwc5m9LNrF6OBS9N6S7DncIYt9VupI5
OCr1uRSqzqaBMFDClTTH+dAx3b6J1KFB0UiHP3FeTalFh8L3NE+dN9apNAgkUWv/
v4oo/6dkRu3NZse2RAo/o2X5r40qk/lhydQRZTSTFsiuH3VUWVsgmqAHnHW7pMMw
8FAlKhyRSFnhbW7re0jj8XMIO7G5yjQKQCnYuPdXbx++bP1PzsEWDv9j/sph5arc
osdo6tEXklWHED17MEPIton1+NRfsU0peEVggQXlwdTcZN/h7FeCZ56dcwCWdCpS
lv6CcWzRXSNUyJpKa9qfIqBX/monjy7w5IHmhvLwAYI6IoT11h1QDEfGfhrwWPwO
jnXsaYm5E7wv8w69PxMbOJbMpWSg8L7xW3LXKR1VwXggUC1+b3y67E5Ggi1hf0lf
TnTMpL2ClAkQ2a0qGAV0dMsNxRAAsuW1aLh+hgydW+iH6DmdQRMESsB1kE02kO14
62TAQaziIAvNoxw5h48xvyEnrDA8d+9IDMyxdrLmAbndUlSveMa9+EPiGHwr6VTy
```

```
FL8nA5F7DcFi4mjEyGKe18JcaAlYUtvHgWH6EjiX2iSXpsrJFEhtfFNolZ5sp9LF
I6hOBihSJxZK4sbMR7Q6IkDuAVpTFLiejBRlsXpFvTGL6040CtxbL5cqkVMYP38r
FMTuc3pGGJA4wb5EC1dGjUi6XjbYH7kuCAFyXqV9eQQP61x7K9W8qnXW+weCIMKf
SX7AcCtH1jXBAM6lqpPrh6amc+/rbg2eNA7DmgJnEY4apIcDB/b4khRMga2ozeGW
WyIvOaVvR2R7ALQ+Rgut85cM+4+Vl2PHmOzW/yYdHVb5REQItFR5COb/mGUqYhkC
tiV3nXo/K0uOQKu5SBbNzLuNvuwdn+Eimxjl18VnrGG7sjtUa0MLmtr62GiEVrhr
Dqa/biHp8LdWkAQjLZ4aTRh2XZiggaVFZHmkw3ILPyKKM21UXdM0YRk3TGVK8ODQ
y58ebPS4v9yYT9gUA9UDkDYeGcF2qjoDPVNvcG6H8jCSsPRl1KZwtqqITCOSAIAP
I4Nu97kO6nbOyQpYlwjd1MhvVXnP66mHSvmqaxbNGX1mF9B/yERkBkooNZrKuJSv
BTC2J1qJBHIEGAEKACYCGwIWIQQL4zJ11OyVPHn4EQfZrSoYBXR0ywUCaW6EcgUJ
CaGkeQJAwXQgBBkBCgAdFiEEthUnfoEIffdcgYM7bljekB8AGu8FAmOVdskACgkQ
bljekB8AGu+4GxAA6VAU9eHHrCE7jLDlmLMhLTZgeFJUxVgpLZP2M8gqLWrGcSIT
D14ofRtbQ6PRMSFk/IzY0kgPsNFYrmvz4qwy7yPCbV9SYit58DHp8JV8JZEQJPUy
VsveOGHoBxf+crZ7cGfy+ATGHNzM7cmW8PrA8JzQWSmJjvIMH2KG7X6BsTkeDp29
+221FcOtsR0CJ9s2n0pjEOKDF2U5vRLXiGMIHKwtYflK7+SFwiWR66plwjxOYOV2
dRvY6pMJWkWFo3Cw9KscHOZvSzaxejgUvTekuw53CGLfVbqSOTgq9bkUqs6mgTBQ
wpU0x/nQMd2+idShQdFIhz9xXk2pRYfC9zRPnTfWqTQIJFFr/7+KKP+nZEbtzWbH
tkQKP6Nl+a+NKpP5YcnUEWU0kxbIrh91VFlbIJqgB5x1u6TDMPBQJSockUhZ4W1u
63tI4/FzCDuxuco0CkAp2Lj3V28fvmz9T87BFg7/Y/7KYeWq3KLHaOrRF5JVhxA9
ezBDyLaJ9fjUX7FNKXhFYIEF5cHU3GTf4exXgmeenXMAlnQqUpb+gnFs0V0jVMia
SmvanyKgV/5qJ48u8OSB5oby8AGCOiKE9dYdUAxHxn4a8Fj8Do517GmJuRO8L/MO
vT8TGziWzKVkoPC+8Vty1ykdVcF4IFAtfm98uuxORoItYX9JX050zKS9gpQJENmt
KhgFdHTLozcQAJCdYMPo1SPgxxRI40BCD4ekuA6ZMzdTuuso9XTTLf3moh+eu0Y+
Pv6ISOFI93g09V34H3XnuaOnA16DyApO6mCPZ58DunjRRYHv4wxbeff6vw9twPQb
bcdNgI2ZXSKjUef4FygkpUhy6075m6zoUfcsR/22Ggzd8Qw3chfoqiBZs7SlGwEH
sxkB+dJvgCMwYaLGqS9lNbHSEFnETb0ihFSHKDrK6pYrFnfsuArJN5owuTN+dzim
8g0agB3Pzs4A6+9Hpt9sN78VEY+ZzFhQD8knEgqSNbFIaNxsG5/L8/RcXuNyPvvz
5MIkhfacOX37YFfAhCORfN3PaVWYQ7oDnv8Nh69i9SkxXKUcwX0/iYaim8pwwM6a
HXhTlqBgKR5b5Hcx6Cwx1GTRUpdbJYThmz7LQgzjBIhmoGbxR3vyJjSyPGBZa3Ih
jAETYzMt7j0qGPiJUx7kcHeK7apAH1xB4DnfuPLSC1GjWptb31A22wjWy/75SJk8
0FoyYKPj5RqiQF+5CcSLRwTJEEi+9ggM8mUJiFNUOtfZgATgmCyEKG4fL1N4iyrj
fBha11/EO3LcH4t0rj5i8SkT+w8A0bgpOliLhnKpFQMy+cRewkzNIS2ynGlHqnKA
biuBEr1ji0iHNHczTMuqSYwraVHis+098D+1me5+9SaDp8yEIKgVuTUAuQINBGOV
dwcBEADKRVXM2aPWTzbIFu/dVFuaOsJ653dZyDvkuK3N+6tkxM/ChhwrsaOTcH32
drei59OASOeKW2qYNe2+2tIT5Gt+xxZC9vJ+FcjHcdArAJt0BH1LHSuscCldkrnG
HlXK1dbqXDeD3Vw+7LtSO/wOAllxSRtOO8ydGfPZSWYkBIgKu4IbxKIpwtLJvtGu
mqP6HKgoFgijNcgjzssyV5Xka79vUkZ2M4jEfqtlNMfcYPV5ehjMFJhcGyXPZzkV
byx+321Ov3bvBIKAGK4JRThWRv00qJUhlkUjImr89CUg/4vqxDeGiE+FnwXvi/Vs
tQJqzcYgDvsypsx0LtyUsQ53K/svoXHfltIKsNdBaCPY1i3C+5EfumtIjo7iUl0q
Katpdw2n4j8bQhZq+AhkIcwRzOnuP7/zTXdxdF6VWyekUKYe5ZH738/Ktpb1JR4X
sydKdcV5oX3/Qh43VTIZgJs+fopgQF5NpqLbTXym9KRRHp9cWNyv1DwOIPqD7Eo+
PeTLLyIPXiDAnAt/89EhFDFh/7Lcsicz6n6cnxsGXk3CI/b4NqPkZxr5nC1WAPxB
k6D5PdOZL/3Cu96MvG2cjgdMzIO0/T2loC1DtLKo6qNDMN5JxckoN6EV5XIpT0L3
z3V9Cb8C6XPQ1pa9ftRpAlydPJVroCXPegiULqSkW9SGwtKFFwARAQABiQI8BBgB
CgAmFiEEC+MydddMlTx5+BEH2a0qGAV0dMsFAmOVdwcCGwwFCQICKQAACgkQ2a0q
GAV0dMsMLQ/8CA5hHB3RZchUU6JlAY23itrs3GSIZJIC+fJXOt1D5XuYidhQb5mE
vtK0Eud7FRLHHJ+Sn5DsE68ipZb1m5dCM7oqn2sMQC3H1qX2SR80zEpwx34F7zjs
hTEK7qkQwBwK+zzQtAp209LBBdmRc2nfDPXy6rZF1MeJoBAjRieAlaoO+SmPLlM7
/wZkcBH+fOxU+f4gCHneTIxQdlRQ1/RgAXL002x8QmSMZ8Zd3LNeUsU9JQKH2drn
lOpFmFmUlxXmhDP+MpjycLNppldAJGyYIS1wp/TMd51+jU2WfR2q267vkc1t/x4N
w2XdCfb5KPD2nz9JdTVrYMl++Aqr04I808V056q0V83HMyu34XAtruItKaz7V13M
```

```
sou0sKaC4pGmQixRYVJjsfpkuU47YBchWQTcZzu4gMJM33E3AvbD6FmhZokH+quR
CGMxvlkKZ3I/NJsZdx1jj+qBY/4WSUWLHinjJLycy06PO6xXTOgrDKy8BE0envml
jNG7TxYpda11QAjTo7MtG8U1fDiRH8sZ2Jso75wbwdJHYmlRA3VYcYEqsywF+JoF
2uPf1ejP7szK/p9/wNSXZOFen1ZvEuXBSF8aFGrd3AeMRmk/Tlg96NTT6+nkvVaN
BLCZUZfNUhuvH0vkLZNn19u0+vGBuAGt8zH2cVLFbhS7U7kG0QvnyaSJAjwEGAEK
ACYCGwwWIQQL4zJ110yVPHn4EQfZrSoYBXR0ywUCZYXLqgUJA/J9aAAKCRDZrSoY
BXR0y6DtEACbExPX5JXm+2MzxLj/Rid48fDxnVpu5/vFDXa9uJEuoDl1VHz2a1mu
UqNtYXCMGW+D2Q6h0lUkJ0XIioUpxu4KZWDzRmePhVmy6/5gUWGq45FRS1Lgj3sU
iI8VdSkhiZ1rPWE1QCmQM++RqQjC+ow+m4gjkAkPOsKjnVr2SGUfGL7q68UeuAAQ
dK427GhvHDTlF9XMAH3S8LYX+uHUvtpt5ajRgx5D+XjS9ESMQuYF2BKdJb5pTB92
bjuVq3x+9z0w4PPW8V3roaxZkhMH+MVputN6IvP9qkL0ff+429KPre9TULlq1iBT
jWVloGQVBrKPrsk914GOGKihkd89kMyh2JyejO0r0mKaku0bXMXAlxdBxjXxEvHA
ILSlaRbW7UsF16fDvKC8o5lSDvzY1KWyzhnRSLGq7HsyxVfX9n6metyd+IclEoPC
62FVdXwOQyVKXm21+IS6XZNDDLYBuNwcCiBpSU6S+UZEP7kFN2FtpGoq8u6PtYpg
w9E0c0MbXY1TiOb55UzXHyWSCHy8eHawAQNdjr6V93Xz1etkfvAQJSRU+N9AhugC
EiSAaiURb7opJp2RlLxB5jlLPxS222l0z5CnO7UdliMJY0bzFfkJPOHFnF+mHITL
JTLogzNZrZUGYoN+FhxkhPswlp1wLJSB5UsKMe6n3TtCs9l53EmyAIkCPAQYAQoA
JgIbDBYhBAvjMnXXTJU8efgRB9mtKhgFdHTLBQJnhEEuBQkF0/ICAAoJENmtKhgF
dHTLJtUP/1dPcTrrUm1lOOCkbq3ZxirJOegxZal7YxmCagz6zg6V9zPCA39jLkyD
dsnzzL/fOGwGzgPUS+xGhlZUFFvUFj1ETLGShBUJY4X8USTvSaBg2CO7kQ50hIul
o1b4FJOToiIdChCOIxHGiws73iBHLMK1Q0gJ/u5xr3sD/Ea8e6jFDbywJbcYjMnx
l53mr0bN7H7rM3W/S002PJQlNMa7JWl6673wpGkW7JBYt0t9SPtWQf3sLgIR6n3N
d3kQYz8EDmpyRXqQMz+BheNX1UDfRrbLAEwg/nVejhYWI31iWziHD79u+8ArxiBb
nDPV0Y25AiGBUBOVvtTv2nLdzm49bAfdAvnFToAp4N+2Knzm2CxdfUROPmuzVAIf
xsR4/vE8VMr762llsnzakzA4xzEoTQ4xtKksxV+738lKZnwH8/umne/GpLPSzH/v
OohpLwYVxbO3M1b+D1lmFdflkoTVNmPcjGh986VlGkRHQdOxymy72uGNYnKllcfs
+sbNz77Km03ov5qF4mZO6IaWwZtIbSZZIttgEWKxl0iusOUULQmaAVtAxI+STQY5
Cul2wFt1RbgANSecC0ts9HiEeWnT/v0sUFG06vmRYjhnQT9YP+/N3u8xHcgNGHwN
ilPTnS87qVIsf8fiWHMZq2V53UjDm3dCIljocF5YGq2ySTEXjjkYiQI8BBgBCgAm
AhsMFiEEC+MydddMlTx5+BEH2a0qGAV0dMsFAmluhHIFCQmhpDsACgkQ2a0qGAV0
dMs5ag//V7mahuI9VikX1lG599GSjm1cXeDrZmGtamz9hYE7EzGlyxf5WQn83Sxu
2kNe9UbaLcIpKzZ9CWcb7NfC6odUIzJsO21qSKNuoP7FV9472JTNFMhaj1+o1P8b
v1nGOU4H9IH/y0Wqkuteef3t4le0iCtePMU3vXgfUFAmm/VF/tgmJQb2hLkXFNCQ
QwK//jjm9QHajaU2wEpvskarsRGfGBintBbUqUUxyPgirqAC8dFAWrjefmGePWEW
JfRLql39II+xbjEzADBB8yqnxLuaUQfVF7t2oAOfp09dP++bCua7FO1WfuFoa1Yw
um84/XszGgkxDRFkBxabpMCurPWgyhkGto4TqKdcWTMl4S1rTxt4Bm/6fqMn+04z
/AiXgU/iIbEKhvQUxsTxyNBLPFIRD7VmrqrOr3wkHgGHJgP0656ABq5J+T4mtFhX
uL0U9nGSm5lV9Hl1ana+Kcs4DqzgQO3QJk/SoulBDZ6at0pK2Y016JrvY51fYpx6
a/IOaXE2JRi8itVlpciDTjSOJs4eNlUx0C3b/LdzU5LZqHtBea1xWxlKPn0VFjYr
YNML6o5B6AL05AcySZj8kNZTWE2ZfOR2usLdiB8b+u2F7WGXZBiQYSnr75WsN94S
nYwO46v5SlzGqTwEN/UBfTkKPklVNgeOMn7fgjziEtMmtBlb7ws=
=ijsc
-----END PGP PUBLIC KEY BLOCK-----
```

D.1.2. Core Team Secretary <core-secretary@FreeBSD.org>

```
pub   rsa4096/4D632518C3546B05 2024-02-17 [SC] [expires: 2028-02-08]
      Key fingerprint = 1A23 6A92 528D 00DD 7965  76FE 4D63 2518 C354 6B05
uid                              FreeBSD Core Team Secretary <core-
```

```
secretary@FreeBSD.org>
sub   rsa4096/CABFDE12CA516ED2 2024-02-17 [E] [expires: 2028-02-08]
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGXQ1o8BEAC+Rcg8cmVxuP17Vu+q5KgCx/XiulQuqKXAqqBlYCH2jqk6DINP
yFrREGBhzd/qNmlAYEahQ4Zgl0bUZNTrZVDyzicOvPP0jH+KSTQwRs7NOawEdlVO
cyHrwDCPEqf5ZzD4NhfTriEOw+j0pEH/onitUGvoQRtx15xWyaJQxDEBMTYMLewE
86D1bltwnTNczE3UZb7oQLJXkAX5hcLtou70XJGgZITvJkK+kp/xot2eFjnqRz/u
WeXnKhYAmC07EKwZ1uw047eHKwMMRBYqzApLwoQtfE430Kxf2q8de64x8zDbi6YM
1J4r8OAxOtHVyfJ0j7Q23DEZz0VVb4b1Tx5OG2Re/KSNvqI0awJO4TcRmOR88OyY
dzyXgnX6Sa7GVQY1FXvn7vtFuDAt7egZOzeomSHL9bdX07LTQ4UtM88EV9wm3q4q
smoatV9jsvPQ1zxCU3aQD/5eWTJH2/kz1LIuBL/Qi5XQpJn91lBtUWJrCgkHWPGu
f//rnnXmsG7DAChW+yZ7cFO8lfNa8sFhPqSxCYphWmJTrvadyQtDngB8JakWdnmK
pfGS6y5lel+181vw38ZZKt04AKM+nDY8051lBM7Q9Q6kTLI33UZeImndx5xYukVD
kV6aQ31HYfEark15c7iEz+OAcwFnM2ntXMt7kKGd40CqzusiPcQkPqPbAQARAQAB
tDhGcmVlQlNEIENvcmUgVGVhbSBTZWNyZXRhcnkgPGNvcmUtc2VjcmV0YXJ5QEZy
ZWVCU0Qub3JnPokCVwQTAQoAQQIbAwgLCQ0IDAcLAwUVCgkICwUWAwIBAAIeBQIX
gBYhBBojapJSjQDdeWV2/k1jJRjDVGsFBQJnp8PiBQkHeofTAAoJEE1jJRjDVGsF
GMIQALhj+mNpH8OmTFeihQ6t9P8un3llz6Wmqe/Q+ULWeqJvV/uC1J5T9fnoGhwF
MgECuguXYJtoYQ16KXnsSOs1tcqMOK9GtEtFJTGe2DtfLiBednwUvu9j3HmTlwLN
M+7rqkiC0HCg2qSjcMjxbVbA5BSgNkgfyTSO2YdjfaZ+ceiHwo/qa5xWE6i2dMR1
PLGMMHTeELdtdSH6VR9/3h0qt3qzwdMBRCVAQHbim7CqwjUH9jg+IOySXl81jNoB
xuVZR3pKshyY4Oxo1dK+W86Uiff/+c4jCAxokGWIR7C4MkZZWUQqV7920gkZFC/5
qzpT1A1/sFUg8HfFTr0vCoPSVFWn2+Tto3vr78DICVaAf02aYAFlyKK18BMCoHkS
hDDO+/JQZmvHOIgEYK+T2WN1c9gm9IDJzGZuH0X2C/vkREnJKkccJfB4pXuN10wt
fqyP9fn8h6+/t+sv3qNLm4d8fkmLXofulG3WB4i/F+Hip2rjPvvBCF7zl4xy6cJ2
xVY5HUOBTqmIwVhYwUpXaqNoWa/qJBLTuot3z7ciKmKX6Lq+Dze5dzhrPNl/CalC
HBf3miHRK3TZbYLooG3bcEWgxU2BnBi3vl5NpCoUOKkPYRlALXi2TyHmPxO7oT4e
mIzS6BgnXOqO+AXvbKKfayuSePdBqkNMK/SMC4Dylkf6Xj25iQIzBBABCgAdFiEE
EBpxaxYrAOVb7eoFrbv4YQo3ibcFAmbu7x4ACgkQrbv4YQo3ibcr5A//TIcbD/EW
YJz0zrUQdc9xG3UNfU6uHmQzAuUy5ginevyqv0TSso5qvSkOHvdxbi41rfMiB2RJ
V3q0n0PSvHFld89fOTZUMZXaPvozCiBYWScrt+KA/2pq3K8mUumHS+IFtpHLL2Tu
+gI8XCHUFxO9HUThM/rFlIyEdzoctgmqQ7IG4uZG+o2J3w09llhDAUe0vraJK1On
p9yFACnRrhqvl41JeUWcv9MH9JcwHUqtUo9WLTCIb+hkByTOyRfHBYpYw//bdXdf
6rkCwKVWpyMDbk61zq2VsS1kqbP9IH/A8CsBnA6mg+zPq2i7HIFw8Swj4OGJcIvG
a9ubUYJRjDhX/vBpNrtncANZ88FQmA+Maq0vu0LS5IIGyIKkvd1fKIsyvBDDa9kE
nfCW0XMkJA0Gf+kxdB+elXQHBwKO2sr5BiKnJT5lJq3Yu9fxxGBnf93yiN4E9bmF
gG7cZxpyb1Bp76TJhLcANyybOTjCtiNRrgqeaSx9/6hSPfPigGXIne0H2lmJO6oq
jUrsYmFiWvU9sc7AcPVw/eHG8FgW35TuwKX71z8w994iaahUPNcSVyXOUD+QNROv
HhGUXrc81t613rivh22N0NZpNubVatq43KV7+/bnPyWBIi1Awh3vIFsNNSQSrYxF
lUuQaAHQXTeZMZ/7npE4t86seMt0T7BGn765Ag0EZdDWjwEQAL3VwFifpnRCYzQ4
VZ1dAjp8w542iRprqeA+C3tvNbk2OvKpN3DIc49l2bgNZ/VI7/T58lEKrFsgLK4w
AWtv18OV7xuh0AuOblmq5MvcPrUelkPj5HA7M6Ng/rAubuHfwdP/IjIrzzY6+XDh
fP9N5KIv9VRJYT23BbvLPeit4J4tQEB/NpxL3L6zI75qq4R3T/EkHP6Y9Buup9Lr
isJlcxkSK+CyORCgTpEz6fWsXiTDgS7cTaQ969XCygBpj4wRQzwkBdokxo1wsiFt
4zLt07/PrPYjeHltTDkrF9XDNhLNJ5GliHnd01oHg1j5/n+9Osvh1maoFwuBxXTN
nTZ2P+7RKLBaVQVSkUa5KJbXoM3v7bMbXM5aLs2XjglrAzrZOV+Y7zOu5UYQdpXd
7x8XAEtEozRJzt7dosQIhKx9h4LlltFFulDUpf5VCvcUEoAzMbiX0aju9n5Rwsql
DHatU9Mm3W8OdFXj1foIXZD+VX2Jp9DgxPLoAJ0CWhPXJ8f+WFSJZCjWoPJEJKWY
```

```
0EOxiXyAuvAyniAZAC/eKZkfGckXmu7edRgYbRTTwPmZ/axa/k9aHsHLwmbxuZo/
xxQXzqU70EleHZ31A3mOqsC1epjN6dn4AFKgvVesP/K/fbvSsfsfiABS2A/68ne4
zJG73Tnk4L6J79vrXc2iMJbmdg3ZABEBAAGJAjwEGAEKACYCGwwWIQQaI2qSUo0A
3Xlldv5NYyUYw1RrBQUCZ6fEFQUJB3qIBgAKCRBNYyUYw1RrBQd9EACMVckxzy4w
aUGlERguJ+kslS8MkMjNqnfDPLRDVxbUxNvbbhw7/u9blM65DCGCeNLc2n4oiu5C
E3I095AKmvq/0dOa81mEEdZkC1CVc64bXWbEyz5AtSHUpgdxRso6C+YopndSiz1T
WcIagQRXfWaw3FBWPooA87gmibmSmCegCtqx+uyc5QxX2eFI8mRK7vlfnGpYKHs0
D1/yUSGQOwoNRJ5FYm+ynfDE3FzHEQL7lv1vpv1k3xNKfBziMMg4IMEBKNHV4VKN
qJpa8UCodeTGSWQdNnCeCWPsxz5oQjCcVH9Z3e8sMpWLHhRcBZzSwXUOws2GbRMH
xHwrfrpHJcrBhe64pgfGOvZLUJ9BDs+8egTHsqRFacipbtTR+hhVhuJEHDaQQWuM
8IRHj9HIuTAczET8JTDHTMIoo8DZdOtiW/YgqCDwYghkI77d12oNQqYoeJ2HiqbK
cGzwCpsR0A+p/iOAxJG13tsxqZV8TQ8iTokWG6ACtZ7sfeWEhxqMbUKUMgogZn0I
3n1kV+tUZC6BQrIYI7TiKg95wLZsIydeoIsQoNZwvyKAXfVmQ62YjIX8njZwN+07
8/ipUPJxCYa8zL8BZyDmoFJqa3y9z+11+vtiZ9t+aTwGvjpHDwyeCJco7go9cU3m
GRFZYciqIoG4n3tl8Pob15vFlVqk47rRqg==
=8TzT
-----END PGP PUBLIC KEY BLOCK-----
```

D.1.3. Ports Management Team Secretary <portmgr-secretary@FreeBSD.org>

```
pub   ed25519/E3C401F60D709D59 2023-03-06 [SC] [expires: 2027-03-05]
      Key fingerprint = BED4 A1D3 6555 B681 2E9F  ABDA E3C4 01F6 0D70 9D59
uid                            FreeBSD Ports Management Team Secretary <portmgr-
secretary@FreeBSD.org>
sub   cv25519/2C92B55E27A641C3 2023-03-06 [E] [expires: 2027-03-05]
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mDMEZAXJvxYJKwYBBAHaRw8BAQdASFAC20WL3R1T6uNyGMZbfJCxDkcP4C5vi3Op
tcZ2fbq0R0ZyZWVCU0QgUG9ydHMgTWFuYWdlbWVudCBUZWFtIFNlY3JldGFyeSA8
cG9ydG1nci1zZWNyZXRhcnlARnJlZUJTRC5vcmc+iJYEExYKAD4WIQS+1KHTZVW2
gS6fq9rjxAH2DXCdWQUCZAXJvwIbAwUJB4TOAAULCQgHAwUVCgkICwUWAwIBAAIe
BQIXgAAKCRDjxAH2DXCdWYN1AP43TjyfZtZ3DLYT++g0+SuPsoO/3yWVybA+UmFL
zb8MngEA+LLNUfvEwCuXS/soh+ww5bpfmi3UUmeGiQEAXug3iA+JATMEEAEKAB0W
IQT7N0XIbxXo7ayBMvzYKU7Du8TX1QUCZAXLkwAKCRDYKU7Du8TX1XHMB/9R1MX4
6zMgpKqPPt76GOI+eGEdBK6bY8aJZjQGdqTh9f6VtXVoTGIG7cvhc9X8tDBoB0PT
2KZWheF51AV1+NHU4HwLAQ1BMebrFvWSfkw4xg4fBGwDhz9/GN85No+Js772V5ey
8lRiL6meRVWxMlLyWcxGd8JjcC5yX/iAUQ3SBGCLqW7unWjjg7CTd+AMBwcqPGrv
ax8q6eFVguJcHJAjMnKf6HAy4cpK3s+uMoUBCGnszSN12B3ysKfyC4pNO/pix5tA
Q5v8aRqTeFPh5zmNhWo0KGPzplTPqRQSHDl7GDQC8Ru3MhzFkeWzHsexjZVwS6W2
DPcYpuuAsA0XOZIZiQIzBBABCgAdFiEEBpxaxYrAOVb7eoFrbv4YQo3ibcFAmQF
0u0ACgkQrbv4YQo3ibccwg/9F2Xuic3nhKxRbB3mJeDo6SYQETa/Gh1qQ34+8zlt
8UMazOx67gnYQfy+pXjro6eQ2up0a4eUYezcNOudqAQD21nRz3HA6EQVNcE/TzEA
xl5CJntTaLOt7S+EDXFW5BuQIvhhoMGgm8+WNVgA0EJ7tfL0OcYBSvr19fqwChEn
9c14cSk6mgHSsleP5NvskYN053pxHwy0LTSb8YBBv52th37t/CRFC1363rS5q+D7
JixFopd1O5pKpA5ipvE4gGgRjPtwjx0SjjepwK/3fuhEJQQyKzTIKlMfu2Dj/iR2
Li1Sfccau5LQXOj9fUITU3u1YG7yrm8VGzT7ao4d+KRwgMLjd2pLqiGIbbJwGBiP
FRmtilWQoeIlmSlFX4obAA517DOK0pW1mH8+eEn4EJd3SekT3yzFyKTASv0J48Z8
```

```
3F928xg+eZvHxVC0t1J+J5IG0gt3EEncuWKIPQGR7PiQbti6R3FQVTz6WfMWOebP
Qi0E9F/Aqakr6Vj2sKGrDq+ebpaF5G8Yw1YrUl2IDiPzkCegp3ZbI0wh11Xvzhi8
LXPQGK4jBQas4G8cegfitzmtdGRHYrbMv0R9I4mvaL+WlOuD2AvyVG28lguqVhnN
AZP+ohdquYyX2CNCVvbKWAtXo6Ur0vWG8BL8m6defAtEkIwVBALaOHQOSI3aNUz4
lwy4OARkBcm/EgorBgEEAZdVAQUBAQdAsefmSfxEOdOr02+K/6noYCuJ1FeAWVz6
jFYQ+9w6jggDAQgHiH4EGBYKACYWIQS+1KHTZVW2gS6fq9rjxAH2DXCdWQUCZAXJ
vwIbDAUJB4TOAAAKCRDjxAH2DXCdWRl4AP9h5ot212BK29S6ZcMBhHvmtF5PG1oD
c7LnZycSRmbFiwEAndCMpAGOhDW8iVgDd0wLQq/ZMPe+xccfG1b3zFH2EgE=
=iiAT
-----END PGP PUBLIC KEY BLOCK-----
```

D.1.4. `doceng-secretary@FreeBSD.org`

```
pub   rsa2048/E1C03580AEB45E58 2019-10-31 [SC] [expires: 2022-10-30]
      Key fingerprint = F24D 7B32 B864 625E 5541  A0E4 E1C0 3580 AEB4 5E58
uid                            FreeBSD Doceng Team Secretary <doceng-
secretary@freebsd.org>
sub   rsa2048/9EA8D713509472FC 2019-10-31 [E] [expires: 2022-10-30]
```

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBF27FFcBCADeoSsIgyQUY8vREwkTikwFFlNg31MVy5s/Nq1cNK1PRfRMnprS
yfB62KqbYuz16bmQKaA9zHN4FGfiTvR6tl66LVHm1s/5HPiLv8sP14GsruLro9zN
v72dO7a9i68bMw+jarPOnu9dGiDFEI0dACOkdCGEYKEUapQeNpmWRrQ46BeXyFwF
JcNx76bJJUkwk6fWC0W63D762e6lCEX6ndoaPjjLBnFvtx13heNGUc8RukBwe2mA
U5pSGHj47J05bdWiRSwZaXa8PcW+20zTWaP755w7zWe4h60GANY7OsT9nuOqsioJ
QonxTrJuZweKRV8fNQ1EfDws3HZr7/7iXvO3ABEBAAG0PEZyZWVCU0QgRG9jZW5n
IFRlYW0gU2VjcmV0YXJ5IDxkb2Nlbmctc2VjcmV0YXJ5QGZyZWVic2Qub3JnPokB
VAQTAQoAPhYhBPJNezK4ZGJeVUGg5OHANYCutF5YBQJduxRXAhsDBQkFo5qABQsJ
CAcDBRUKCQgLBRYDAgEAAh4BAheAAAoJEOHANYCutF5YB2IIALw+EPYmOz9qlqIn
oTFmk/5MrcdzC5iLEfxubbF6TopDWsWPiOh5mAuvfEmROSGf6ctvdYe9UtQV3VNY
KeeyskeFrIBOFo2KG/dFqKPAWef6IfhbW3HWDWo5uOBg01jHzQ/pB1n6SMKiXfsM
idL9wN+UQKxF3Y7S/bVrZTV0isRUolO9+8kQeSYT/NMojVM0H2fWrTP/TaNEW4fY
JBDAl5hsktzdl8sdbNqdC0GiX3xb4GvgVzGGQELagsxjfuXk6PfOyn6Wx2d+yRcI
FrKojmhihBp5VGFQkntBIXQkaW0xhW+WBGxwXdaAl0drQlZ3W+edgdOl705x73kf
Uw3Fh2a5AQ0EXbsUVwEIANEPAsltM4vFj2pi5xEuHEcZIrIX/ZJhoaBtZkqvkB+H
4pu3/eQHK5hg0Dw12ugffPMz8mi57iGNI9TXd8ZYMJxAdvEZSDHCKZTX9G+FcxWa
/AzKNiG25uSISzz7rMB/lV1gofCdGtpHFRFTiNxFcoacugTdlYDiscgJZMJSg/hC
GXBdEKXR5WRAgAGandcL8llCToOt1lZEOkd5vJM861w6evgDhAZ2HGhRuG8/NDxG
r4UtlnYGUCFof/Q4oPNbDJzmZXF+8OQyTNcEpVD3leEOWG1Uv5XWS2XKVHcHZZ++
ISo/B5Q6Oi3SJFCVV9f+g09YF+PgfP/mVMBgif2fT20AEQEAAYkBPAQYAQoAJhYh
BPJNezK4ZGJeVUGg5OHANYCutF5YBQJduxRXAhsMBQkFo5qAAAoJEOHANYCutF5Y
kecIAMTh2VHQqjXHTszQMsy3NjiTVVITI3z+pzY0u2EYmLytXQ2pZMzLHMcklmub
5po0X4EvL6bZiJcLMI2mSrOs0Gp8P3hyMI40IkqoLMp7VA2LFlPgIJ7K5W4oVwf8
khY6lw7qg2l69APm/MM3xAyiL4p6MU8tpvWg5AncZ6lxyy27rxVflzEtCrKQuG/a
oVaOlMjH3uxvOK6IIxlhvWD0nKs/e2h2HIAZ+ILE6ytS5ZEg2GXuigoQZdEnv71L
xyvE9JANwGZLkDxnS5pgN2ikfkQYlFpJEkrNTQleCOHIIIp8vgJngEaP51xOIbQM
CiG/y3cmKQ/ZfH7BBvlZVtZKQsI=
=MQKT
-----END PGP PUBLIC KEY BLOCK-----
```

FreeBSD Glossary

This glossary contains terms and acronyms used within the FreeBSD community and documentation.

A

ACL

See Access Control List.

ACPI

See Advanced Configuration and Power Interface.

AMD

See Automatic Mount Daemon.

AML

See ACPI Machine Language.

API

See Application Programming Interface.

APIC

See Advanced Programmable Interrupt Controller.

APM

See Advanced Power Management.

APOP

See Authenticated Post Office Protocol.

ASL

See ACPI Source Language.

ATA

See Advanced Technology Attachment.

ATM

See Asynchronous Transfer Mode.

ACPI Machine Language

Pseudocode, interpreted by a virtual machine within an ACPI-compliant operating system, providing a layer between the underlying hardware and the documented interface presented to the OS.

ACPI Source Language

The programming language AML is written in.

Access Control List

A list of permissions attached to an object, usually either a file or a network device.

Advanced Configuration and Power Interface

A specification which provides an abstraction of the interface the hardware presents to the operating system, so that the operating system should need to know nothing about the underlying hardware to make the most of it. ACPI evolves and supersedes the functionality provided previously by APM, PNPBIOS and other technologies, and provides facilities for controlling power consumption, machine suspension, device enabling and disabling, etc.

Application Programming Interface

A set of procedures, protocols and tools that specify the canonical interaction of one or more program parts; how, when and why they do work together, and what data they share or operate on.

Advanced Power Management

An API enabling the operating system to work in conjunction with the BIOS in order to achieve power management. APM has been superseded by the much more generic and powerful ACPI specification for most applications.

Advanced Programmable Interrupt Controller

Advanced Technology Attachment

Asynchronous Transfer Mode

Authenticated Post Office Protocol

Automatic Mount Daemon

A daemon that automatically mounts a filesystem when a file or directory within that filesystem is accessed.

B

BAR

See Base Address Register.

BIND

See Berkeley Internet Name Domain.

BIOS

See Basic Input/Output System.

BSD

See Berkeley Software Distribution.

Base Address Register

The registers that determine which address range a PCI device will respond to.

Basic Input/Output System

The definition of BIOS depends a bit on the context. Some people refer to it as the ROM chip with a basic set of routines to provide an interface between software and hardware. Others refer to it as the set of routines contained in the chip that help in bootstrapping the system. Some might also refer to it as the screen used to configure the bootstrapping process. The BIOS is PC-specific but other systems have something similar.

Berkeley Internet Name Domain

An implementation of the DNS protocols.

Berkeley Software Distribution

This is the name that the Computer Systems Research Group (CSRG) at The University of California at Berkeley gave to their improvements and modifications to AT&T's 32V UNIX®. FreeBSD is a descendant of the CSRG work.

Bikeshed Building

A phenomenon whereby many people will give an opinion on an uncomplicated topic, whilst a complex topic receives little or no discussion. See the FAQ for the origin of the term.

C

CD

See Carrier Detect.

CHAP

See Challenge Handshake Authentication Protocol.

CLIP

See Classical IP over ATM.

COFF

See Common Object File Format.

CPU

See Central Processing Unit.

CTS

See Clear To Send.

Carrier Detect

An RS232C signal indicating that a carrier has been detected.

Central Processing Unit

Also known as the processor. This is the brain of the computer where all calculations take place. There are a number of different architectures with different instruction sets. Among the more well-known are the Intel-x86 and derivatives, Arm, and PowerPC.

Challenge Handshake Authentication Protocol

A method of authenticating a user, based on a secret shared between client and server.

Classical IP over ATM

Clear To Send

An RS232C signal giving the remote system permission to send data.

See Also Request To Send.

Common Object File Format

D

DAC

See Discretionary Access Control.

DDB

See Debugger.

DES

See Data Encryption Standard.

DHCP

See Dynamic Host Configuration Protocol.

DNS

See Domain Name System.

DRM

See Direct Rendering Manager.

DSDT

See Differentiated System Description Table.

DSR

See Data Set Ready.

DTR

See Data Terminal Ready.

DVMRP

See Distance-Vector Multicast Routing Protocol.

Discretionary Access Control

Data Encryption Standard

A method of encrypting information, traditionally used as the method of encryption for UNIX® passwords and the crypt(3) function.

Data Set Ready

An RS232C signal sent from the modem to the computer or terminal indicating a readiness to send and receive data.

See Also Data Terminal Ready.

Data Terminal Ready

An RS232C signal sent from the computer or terminal to the modem indicating a readiness to send and receive data.

Debugger

An interactive in-kernel facility for examining the status of a system, often used after a system has crashed to establish the events surrounding the failure.

Differentiated System Description Table

An ACPI table, supplying basic configuration information about the base system.

Distance-Vector Multicast Routing Protocol

Domain Name System

The system that converts humanly readable hostnames (i.e., mail.example.net) to Internet addresses and vice versa.

Direct Rendering Manager

The drm(7) kernel module allows client applications direct access to the graphical hardware via the Direct Rendering Infrastructure.

Dynamic Host Configuration Protocol

A protocol that dynamically assigns IP addresses to a computer (host) when it requests one from the server. The address assignment is called a “lease”.

E

ECOFF

See Extended COFF.

ELF

See Executable and Linking Format.

ESP

See Encapsulated Security Payload.

Encapsulated Security Payload

Executable and Linking Format

Extended COFF

F

FADT

See Fixed ACPI Description Table.

FAT

See File Allocation Table.

FAT16

See File Allocation Table (16-bit).

FTP

See File Transfer Protocol.

File Allocation Table

File Allocation Table (16-bit)

File Transfer Protocol

A member of the family of high-level protocols implemented on top of TCP which can be used to transfer files over a TCP/IP network.

Fixed ACPI Description Table

G

GUI

See Graphical User Interface.

Giant

The name of a mutual exclusion mechanism (a sleep `mutex`) that protects a large set of kernel resources. Although a simple locking mechanism was adequate in the days where a machine might have only a few dozen processes, one networking card, and certainly only one processor, in current times it is an unacceptable performance bottleneck. FreeBSD developers are actively working to replace it with locks that protect individual resources, which will allow a much greater degree of parallelism for both single-processor and multi-processor machines.

Graphical User Interface

A system where the user and computer interact with graphics.

H

HTML

See HyperText Markup Language.

HUP

See HangUp.

HangUp

HyperText Markup Language

The markup language used to create web pages.

I

I/O

See Input/Output.

IASL

See Intel's ASL compiler.

IMAP

See Internet Message Access Protocol.

IP

See Internet Protocol.

IPFW

See IP Firewall.

IPP

See Internet Printing Protocol.

IPv4

See IP Version 4.

IPv6

See IP Version 6.

ISP

See Internet Service Provider.

IP Firewall

IP Version 4

The IP protocol version 4, which uses 32 bits for addressing. This version is still the most widely used, but it is slowly being replaced with IPv6.

See Also IP Version 6.

IP Version 6

The new IP protocol. Invented because the address space in IPv4 is running out. Uses 128 bits for addressing.

Input/Output

Intel's ASL compiler

Intel's compiler for converting ASL into AML.

Internet Message Access Protocol

A protocol for accessing email messages on a mail server, characterised by the messages usually being kept on the server as opposed to being downloaded to the mail reader client.

See Also Post Office Protocol Version 3.

Internet Printing Protocol

Internet Protocol

The packet transmitting protocol that is the basic protocol on the Internet. Originally developed at the U.S. Department of Defense and an extremely important part of the TCP/IP stack. Without the Internet Protocol, the Internet would not have become what it is today. For more information, see RFC 791.

Internet Service Provider

A company that provides access to the Internet.

K

KAME

Japanese for "turtle", the term KAME is used in computing circles to refer to the KAME Project, who work on an implementation of IPv6.

KDC

See Key Distribution Center.

KLD

See Kernel ld(1).

KMS

See Kernel Mode Setting.

KSE

See Kernel Scheduler Entities.

KVA

See Kernel Virtual Address.

Kbps

See Kilo Bits Per Second.

Setting the display mode in kernel space.

Kernel ld(1)

A method of dynamically loading functionality into a FreeBSD kernel without rebooting the system.

Kernel Scheduler Entities

A kernel-supported threading system. See the KSE Project home page for further details.

Kernel Virtual Address

Key Distribution Center

Kilo Bits Per Second

Used to measure bandwidth (how much data can pass a given point at a specified amount of time). Alternates to the Kilo prefix include Mega, Giga, Tera, and so forth.

L

LAN

See Local Area Network.

LOR

See Lock Order Reversal.

LPD

See Line Printer Daemon.

Line Printer Daemon

Local Area Network

A network used on a local area, e.g. office, home, or so forth.

Lock Order Reversal

The FreeBSD kernel uses a number of resource locks to arbitrate contention for those resources. A run-time lock diagnostic system found in FreeBSD-CURRENT kernels (but removed for releases), called witness(4), detects the potential for deadlocks due to locking errors. (witness(4) is actually slightly conservative, so it is possible to get false positives.) A true positive report indicates that “if you were unlucky, a deadlock would have happened here”.

True positive LORs tend to get fixed quickly, so check https://lists.FreeBSD.org/subscription/freebsd-current and the LORs Seen page before posting to the mailing lists.

M

MAC

See Mandatory Access Control.

MADT

See Multiple APIC Description Table.

MFC

See Merge From Current.

MFH

See Merge From Head.

MFS

See Merge From Stable.

MFV

See Merge From Vendor.

MIT

See Massachusetts Institute of Technology.

MLS

See Multi-Level Security.

MOTD

See Message Of The Day.

MTA

See Mail Transfer Agent.

MUA

See Mail User Agent.

Mail Transfer Agent

An application used to transfer email. An MTA has traditionally been part of the BSD base system. Today Sendmail is included in the base system, but there are many other MTAs, such as postfix, qmail and Exim.

Mail User Agent

An application used by users to display and write email.

Mandatory Access Control

Massachusetts Institute of Technology

Merge From Current

To merge functionality or a patch from the -CURRENT branch to another, most often -STABLE.

Merge From Head

To merge functionality or a patch from a repository HEAD to an earlier branch.

Merge From Stable

In the normal course of FreeBSD development, a change will be committed to the -CURRENT branch for testing before being merged to -STABLE. On rare occasions, a change will go into -STABLE first and then be merged to -CURRENT.

This term is also used when a patch is merged from -STABLE to a security branch.

See Also Merge From Current.

Merge From Vendor

Message Of The Day

A message, usually shown on login, often used to distribute information to users of the system.

Multi-Level Security

Multiple APIC Description Table

N

NAT

See Network Address Translation.

NDISulator

See Project Evil.

NFS

See Network File System.

NTFS

See New Technology File System.

NTP

See Network Time Protocol.

Network Address Translation

A technique where IP packets are rewritten on the way through a gateway, enabling many machines behind the gateway to effectively share a single IP address.

Network File System

New Technology File System

A filesystem developed by Microsoft and available in its “New Technology” operating systems, such as Windows® 2000, Windows NT® and Windows® XP.

Network Time Protocol

A means of synchronizing clocks over a network.

O

OBE

See Overtaken By Events.

ODMR

See On-Demand Mail Relay.

OS

See Operating System.

On-Demand Mail Relay

Operating System

A set of programs, libraries and tools that provide access to the hardware resources of a computer. Operating systems range today from simplistic designs that support only one program running at a time, accessing only one device to fully multi-user, multi-tasking and multi-process systems that can serve thousands of users simultaneously, each of them running dozens of different applications.

Overtaken By Events

Indicates a suggested change (such as a Problem Report or a feature request) which is no longer relevant or applicable due to such things as later changes to FreeBSD, changes in networking standards, the affected hardware having since become obsolete, and so forth.

P

PAE

See Physical Address Extensions.

PAM

See Pluggable Authentication Modules.

PAP

See Password Authentication Protocol.

PC

See Personal Computer.

PCNSFD

See crossref:glossary[pcnfsd-glossary,Personal Computer Network File System Daemon].

PDF

See Portable Document Format.

PID

See Process ID.

POLA

See Principle Of Least Astonishment.

POP

See Post Office Protocol.

POP3

See Post Office Protocol Version 3.

PPD

See PostScript Printer Description.

PPP

See Point-to-Point Protocol.

PPPoA

See PPP over ATM.

PPPoE

See PPP over Ethernet.

PPP over ATM

PPP over Ethernet

PR

See Problem Report.

PXE

See Preboot eXecution Environment.

Password Authentication Protocol

Personal Computer

Personal Computer Network File System Daemon

Physical Address Extensions

A method of enabling access to up to 64 GB of RAM on systems which only physically have a 32-bit wide address space (and would therefore be limited to 4 GB without PAE).

Pluggable Authentication Modules

Point-to-Point Protocol

Pointy Hat

A mythical piece of headgear, much like a dunce cap, awarded to any FreeBSD committer who breaks the build, makes revision numbers go backwards, or creates any other kind of havoc in the source base. Any committer worth his or her salt will soon accumulate a large collection. The usage is (almost always?) humorous.

Portable Document Format

Post Office Protocol

See Also Post Office Protocol Version 3.

Post Office Protocol Version 3

A protocol for accessing email messages on a mail server, characterised by the messages usually being downloaded from the server to the client, as opposed to remaining on the server.

See Also Internet Message Access Protocol.

PostScript Printer Description

Preboot eXecution Environment

Principle Of Least Astonishment

As FreeBSD evolves, changes visible to the user should be kept as unsurprising as possible. For example, arbitrarily rearranging system startup variables in /etc/defaults/rc.conf violates POLA. Developers consider POLA when contemplating user-visible system changes.

PRIME

A method of multiple physical graphics coprocessors coexisting by sharing their direct memory access buffers.

Problem Report

A description of some kind of problem that has been found in either the FreeBSD source or documentation. See Writing FreeBSD Problem Reports.

Process ID

A number, unique to a particular process on a system, which identifies it and allows actions to be taken against it.

Project Evil

The working title for the NDISulator, written by Bill Paul, who named it referring to how awful it is (from a philosophical standpoint) to need to have something like this in the first place. The NDISulator is a special compatibility module to allow Microsoft Windows™ NDIS miniport network drivers to be used with FreeBSD/i386. This is usually the only way to use cards where the driver is closed-source. See src/sys/compat/ndis/subr_ndis.c.

R

RA

See Router Advertisement.

RAID

See Redundant Array of Inexpensive Disks.

RAM

See Random Access Memory.

RD

See Received Data.

RFC

See Request For Comments.

RISC

See Reduced Instruction Set Computer.

RPC

See Remote Procedure Call.

RS232C

See Recommended Standard 232C.

RTS

See Request To Send.

Random Access Memory

Revision Control System

The *Revision Control System (RCS)* is one of the oldest software suites that implement “revision control” for plain files. It allows the storage, retrieval, archival, logging, identification and merging of multiple revisions for each file. RCS consists of many small tools that work together. It lacks some of the features found in more modern revision control systems, like Git, but it is very simple to install, configure, and start using for a small set of files.

See Also Subversion.

Received Data

An RS232C pin or wire that data is received on.

See Also Transmitted Data.

Recommended Standard 232C

A standard for communications between serial devices.

Reduced Instruction Set Computer

An approach to processor design where the operations the hardware can perform are simplified but made as general purpose as possible. This can lead to lower power consumption, fewer transistors and in some cases, better performance and increased code density. Examples of RISC processors include the Alpha, SPARC®, ARM® and PowerPC®.

Redundant Array of Inexpensive Disks

Remote Procedure Call

Request For Comments

A set of documents defining Internet standards, protocols, and so forth. See www.rfc-editor.org.

Also used as a general term when someone has a suggested change and wants feedback.

Request To Send

An RS232C signal requesting that the remote system commences transmission of data.

See Also Clear To Send.

Router Advertisement

S

SCI

See System Control Interrupt.

SCSI

See Small Computer System Interface.

SG

See Signal Ground.

SLAAC

See StateLess Address AutoConfiguration.

SMB

See Server Message Block.

SMP

See Symmetric MultiProcessor.

SMTP

See Simple Mail Transfer Protocol.

SMTP AUTH

See SMTP Authentication.

SSH

See Secure Shell.

STR

See Suspend To RAM.

SVN

See Subversion.

StateLess Address AutoConfiguration

SMTP Authentication

Server Message Block

Signal Ground

An RS232 pin or wire that is the ground reference for the signal.

Simple Mail Transfer Protocol

Secure Shell

Small Computer System Interface

Subversion

Subversion is a version control system currently used by the FreeBSD project.

Suspend To RAM

Symmetric MultiProcessor

System Control Interrupt

T

TCP

See Transmission Control Protocol.

TCP/IP

See crossref:glossary[tcpip-glossary,Transmission Control Protocol/Internet Protocol].

TD

See Transmitted Data.

TFTP

See Trivial FTP.

TGT

See Ticket-Granting Ticket.

TSC

See Time Stamp Counter.

Ticket-Granting Ticket

Time Stamp Counter

A profiling counter internal to modern Pentium® processors that counts core frequency clock ticks.

Transmission Control Protocol

A protocol that sits on top of (e.g.) the IP protocol and guarantees that packets are delivered in a reliable, ordered, fashion.

Transmission Control Protocol/Internet Protocol

The term for the combination of the TCP protocol running over the IP protocol. Much of the Internet runs over TCP/IP.

Transmitted Data

An RS232C pin or wire that data is transmitted on.

See Also Received Data.

Trivial FTP

U

UDP

See User Datagram Protocol.

UFS1

See Unix File System Version 1.

UFS2

See Unix File System Version 2.

UID

See User ID.

URL

See Uniform Resource Locator.

USB

See Universal Serial Bus.

Uniform Resource Locator

A method of locating a resource, such as a document on the Internet and a means to identify that resource.

Unix File System Version 1

The original UNIX® file system, sometimes called the Berkeley Fast File System.

Unix File System Version 2

An extension to UFS1, introduced in FreeBSD 5-CURRENT. UFS2 adds 64 bit block pointers (breaking the 1T barrier), support for extended file storage and other features.

Universal Serial Bus

A hardware standard used to connect a wide variety of computer peripherals to a universal interface.

User ID

A unique number assigned to each user of a computer, by which the resources and permissions assigned to that user can be identified.

User Datagram Protocol

A simple, unreliable datagram protocol which is used for exchanging data on a TCP/IP network. UDP does not provide error checking and correction like TCP.

V

VPN

See Virtual Private Network.

Virtual Private Network

A method of using a public telecommunication such as the Internet, to provide remote access to a localized network, such as a corporate LAN.

Colophon

This book is the combined work of hundreds of contributors to "The FreeBSD Documentation Project". The text is authored in AsciiDoc.